UIUCDCS-R-78-900 UILU-ENG 78 1715

INTELLIGENT MAGENTIC BUBBLE MEMORIES

by

Mario Jino

May 1978

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

INTELLIGENT MAGNETIC BUBBLE MEMORIES

BY

MARIO JINO

Eng. Elet., Instituto Tecnológico de Aeronautica, 1967
M. Cien., Universidade Estadual de Campinas, 1974

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1978

Urbana, Illinois

---

* This work was supported in part by Universidade Estadual de Campinas and Fundacão de Amparo a Pesquisa do Estado de Sao Paulo (Processo Matematica 73/875) and was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, May 1978.

** This work was supported in part by the National Science Foundation under Grant Number MCS-77-27910.

ACKNOWLEDGEMENT

The author expresses his deepest gratitude to Professor Jane W. S. Liu. Her enlightening advice and continuing help throughout this work were instrumental to its conclusion and are most appreciated. He also wishes to thank Professors H. G. Friedman, Jr., D. J. Kuck, D. H. Lawrie and S. R. Ray for their comments and suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

Page

1. INTRODUCTION

The design and implementation of special purpose hardwares to support file processing and data base management functions have received a great deal of attention. Since large associative file processor using disks with search logic attached to read/write heads was first suggested by Slotnick [1], many architectures of logic-per-track storage systems were proposed and studied. Earlier efforts in this area [2-5] were primarily directed towards providing partial associativity at reasonably low cost and high throughput rates. More recently, several systems were designed and implemented to support general data structure and basic data base management functions as well as content addressability. Among these, CASSM [6,7] and RAP [8] are intended to be stand alone data base machines. RARES [9], on the other hand, is an intelligent storage system designed to perform tuple selection and sorting operations, and thus enhance the performance of the query interface to the data base. All three systems were designed to support the relational model. The database computer (DBC), proposed by Hsiao and Kannan [10], is intended to be a back end computer capable of handling very large data bases (10^9 bytes or more) and of supporting several data base models. In addition, it also provides security and clustering mechanisms, making DBC a complete self contained data base management system.

The recent emergence of new memory technologies, such

as magnetic bubble memories, charge-coupled devices and electron beam addressed memories, provides us with more promise of realizing intelligent memories economically. These devices are likely to replace fixed-head disk technology. Moreover, novel memory configurations and operations made possible by these memory techniques allow us new design alternatives. The feasibility of using these three new memory devices for retrieval and update of structured information has been examined in [10].

In this report, we are concerned with the design of intelligent magnetic bubble memories. Such memories may be used to store and maintain structured information in data base machines as suggested in [10]. They may also be used to provide users with large work spaces in which elementary file processing operations may be performed without external intervention or as intelligent circulating memories of the type described in [11]. Figure 1 shows an example of data base machine architecture making use of the type of intelligent memories considered here. The intelligent memory, like in RARES, is intended to be an intelligent storage system to enhance the performance of the query interface to the data base. Queries and data from front end computers are sent to the intelligent memory via the data base interface processor (DIP). Similarly, responses from the data base are sent back under the control of the DIP. For the sake of concreteness, we assume that the data base machine supports directly the relational data model. DIP translates queries into sequences of primitive relational algebraic

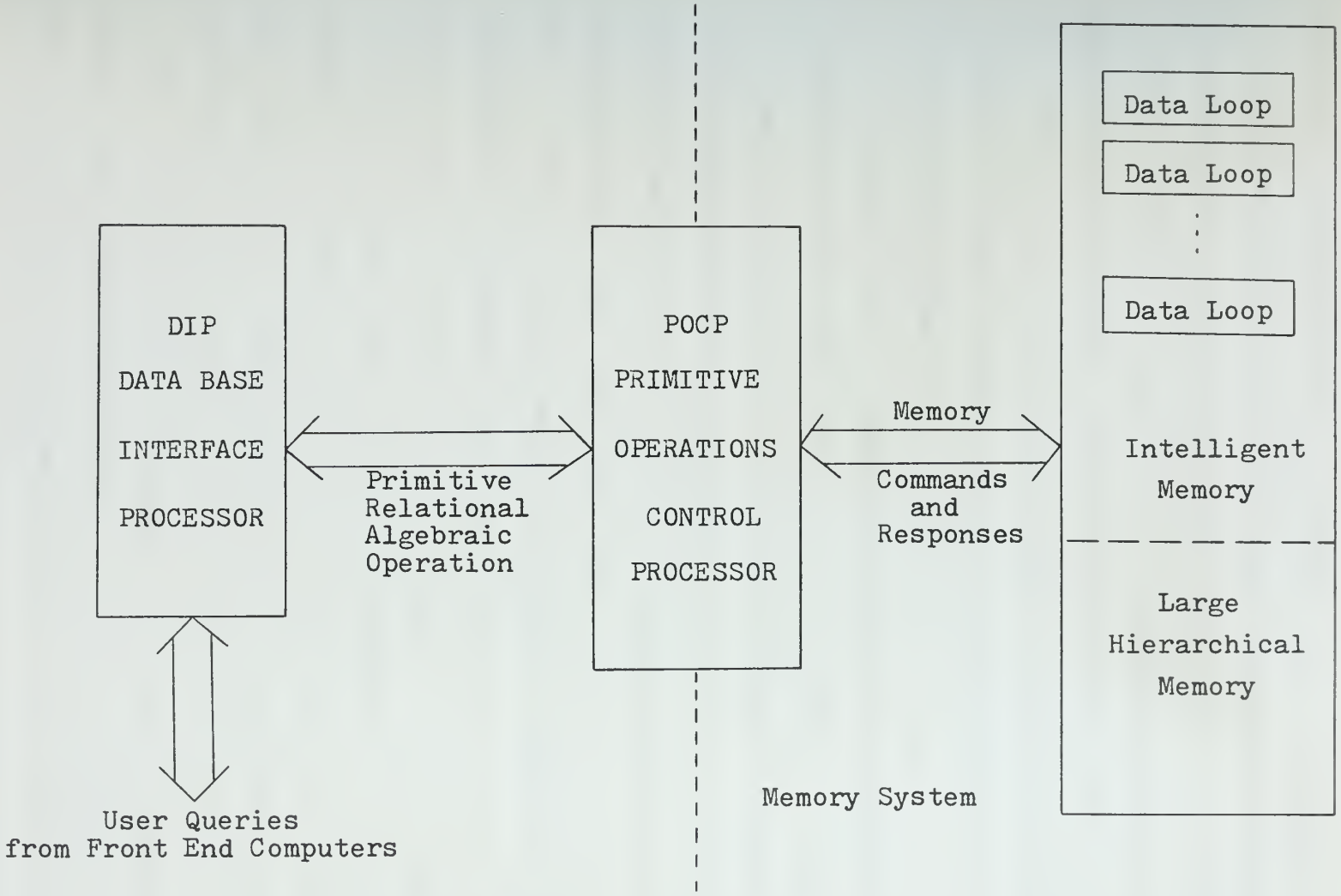


Figure 1. An example of data base machine architecture.

operations (similar to the interface SQUIRAL described in [12]). These relational algebraic operations are in turn translated into sequences of commands to memory modules by the primitive operations control processor (POCP). Memory commands in terms of elementary file processing operations are performed autonomously within the memory modules and monitored by the POCP.

Our purpose here is not to propose hardware configurations of intelligent memory that can be readily implemented with existing technology [13-15]. Rather, it is our intent to explore ways of incorporating the novel bubble chip organizations and bubble movement operation [16-21] in the design of such memories. In particular, we evaluate the performance of various memory organizations and algorithms for elementary file processing operations and basic relational algebraic operations which can be achieved through their use. Earlier efforts in this direction include the studies on dynamic access of an item and data reorganization in magnetic bubble memories [16-19], and the use of N-loop shift registers linked by flow steering switches to sort the N records stored in the N loops [20,21]. The latter idea has been extended to the proposed usage of two-dimensional arrays of shift register loops [22]. Although the discussion throughout this thesis is on magnetic bubble memories, the results in algorithm design and memory organization are clearly applicable in any shift register memories in which the same data rearrangement operations are allowed.

Algorithm design and memory organization are based

on the data rearrangement operations and bubble chip organizations presented in Chapter 2. Also included in Chapter 2 is an introductory description of the relational data model.

Several ways of organizing data in bubble memories are studied in Chapter 3. Retrieval times per word and per page are the parameters used to evaluate the different memory organizations. Performance of hierarchical memory systems using magnetic bubble memories is discussed.

In Chapter 4, the unique features of magnetic bubble memories are used in the design of algorithms for elementary file processing operations such as sorting, merging and clustering. The results of Chapters 3 and 4 are extended in Chapter 5. Algorithms for basic relational algebraic operations are discussed.

Finally, in Chapter 6, areas and topics for further research are presented.

2. BACKGROUND AND BASIC CONCEPTS

Our discussions in subsequent chapters are based on the bubble chip organizations and the models of data rearrangement operations presented here. Also presented here in this chapter are an introductory description of the relational data model and most of the notation used throughout the remainder of this thesis.

2.1. Bubble Chip Organizations

Bubble chip organization has evolved from one single long shift register with one access port per chip to the three basic organizations: major-minor loop, on-chip decoder, and bubble lattice chip organizations. Major-minor loop organization [16,17], shown schematically in Figure 2, is the best known of these three basic organizations. Minor loops are circular shift registers sharing the same shift control. They are shifted simultaneously. For the contents of a bubble to be detected and read out, it has to be transferred to the major loop, and then shifted to the access port.

On-chip decoder organization, also known as self-contained magnetic bubble-domain memory chip [23,24], is shown schematically in Figure 3. In this case, a chip is also composed of circular shift registers, or storage loops, which are shifted simultaneously. Instead of the major loop, the mechanism for sharing I/O is the decoder. The usual mode of operation is to

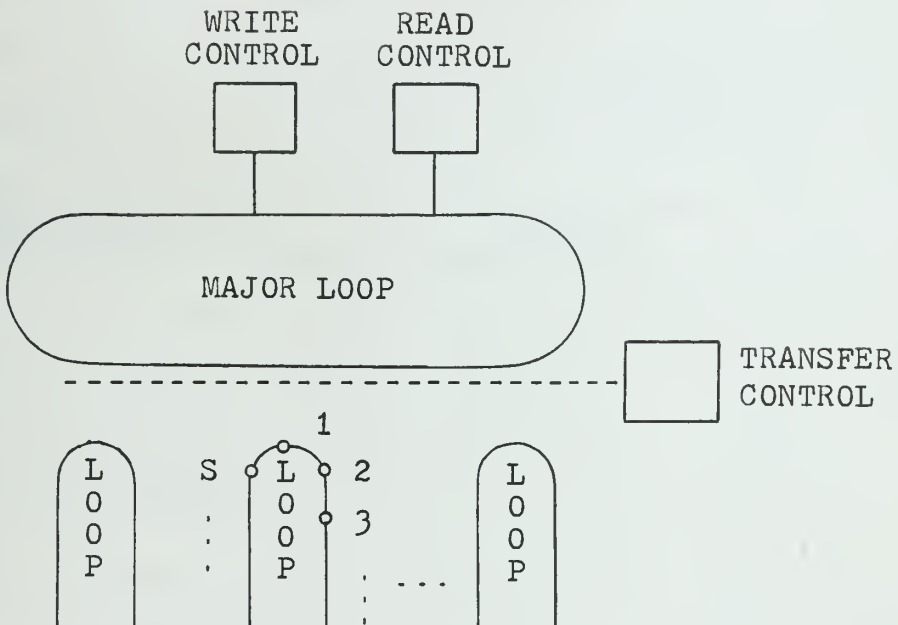


Figure 2. Major-minor loop organization.

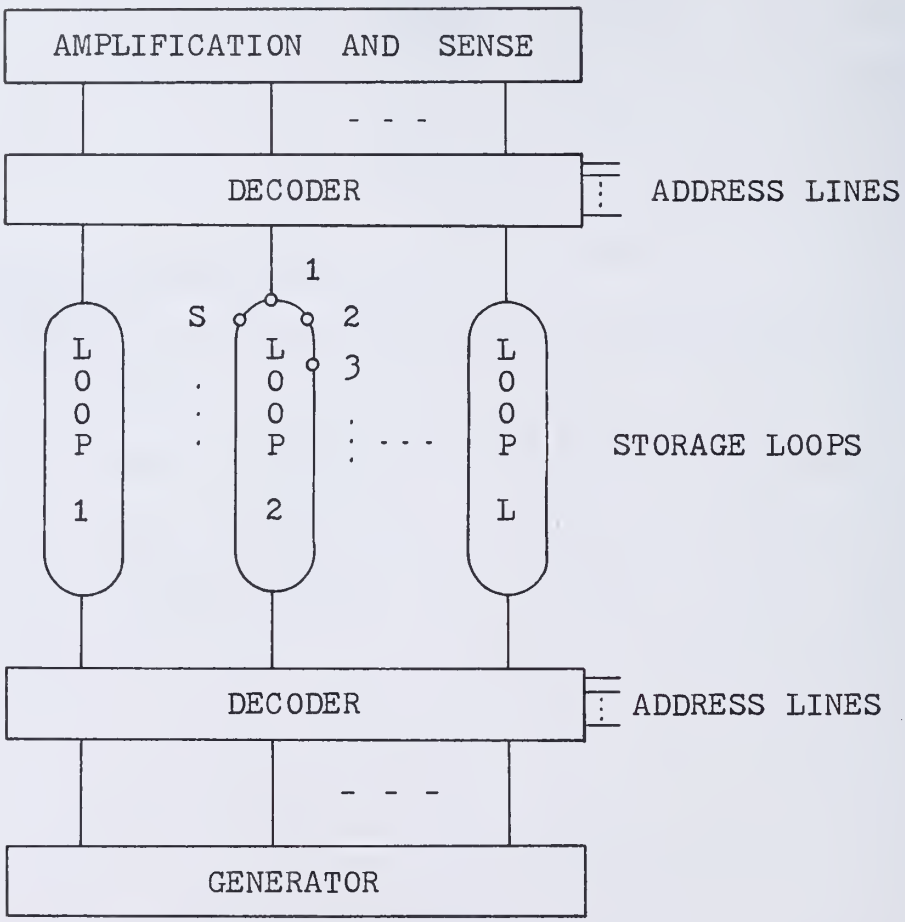


Figure 3. On-chip decoder organization.

shift replicated bubbles from all loops through the decoder. The address lines eliminate all but the bubbles from the selected loop and deliver them to the amplification and sensing device. Bubbles from non-selected loops are lead to bubble collapsers. Bubble splitters duplicate the bubble before they are entered into the decoder. Hence, no information is lost by bubble collapsing. With L storage loops, there are 2L paths in the decoder, two for each loop. When activated, each address line closes half of the output paths. Bubbles in closed paths are diverted to paths leading to bubble collapsers. Therefore, $\log_2 L$ address lines are needed. A bubble is shifted $\log_2 L$ positions from the input to the output of the decoder.

The scheme described above is well suited for the access of one whole storage loop but performs poorly for the sequential access of bubbles from different loops. To make the access of bubbles from different loops more efficient, a stepped control of the address lines is proposed. Initially, the first address line is selected, one propagation time later the first and the second address lines are selected, and so on until all $\log_2 L$ are selected. From then on, all $\log_2 L$ address lines are selected, for as long as it is necessary to output all information. During the first propagation time, L/2 bubbles (from the loops selected by the first address line) propagate through the first portion of their output paths. In the second propagation time, L/4 of these bubbles, selected by the second address line, propagate through the second portion of their

output paths. At the same time, L/2 bubbles, either from the L/2 loops selected in the previous propagation time or from the other L/2 loops, propagate through the first portion of their output paths. After $\log_2 L$ propagation times, there is always one bubble at the output of the decoder, even when accessing bubbles from different loops.

Bubble lattice organization [18,25,26], shown schematically in Figure 4, can be viewed as a variation of the major-minor loop organization. Bubbles are shifted simultaneously left or right in the matrix. Buffer zones are provided to allow this operation without loss of information. The only column that can be shifted up or down is the read/write column (access channel), which is used to bring bubbles to the detector or to enter bubbles.

The following assumptions with regard to bubble chip organizations are made throughout this report: 1) Only one access port is provided to read from and write into a chip one bit at a time, and 2) All shift operations are bidirectional. When it is not necessary to distinguish among them, we refer to minor loop, storage loop, and column specific to the three basic chip organizations simply as storage loops. A chip is composed of L S-bit storage loops, that is, a chip is capable of storing B (= LS) bits of information. Figures 2 - 4 illustrate the convention adopted for the numbering of minor loops, storage loops and columns, and of positions inside them, for the three basic organizations. We define memory module as the indepen-

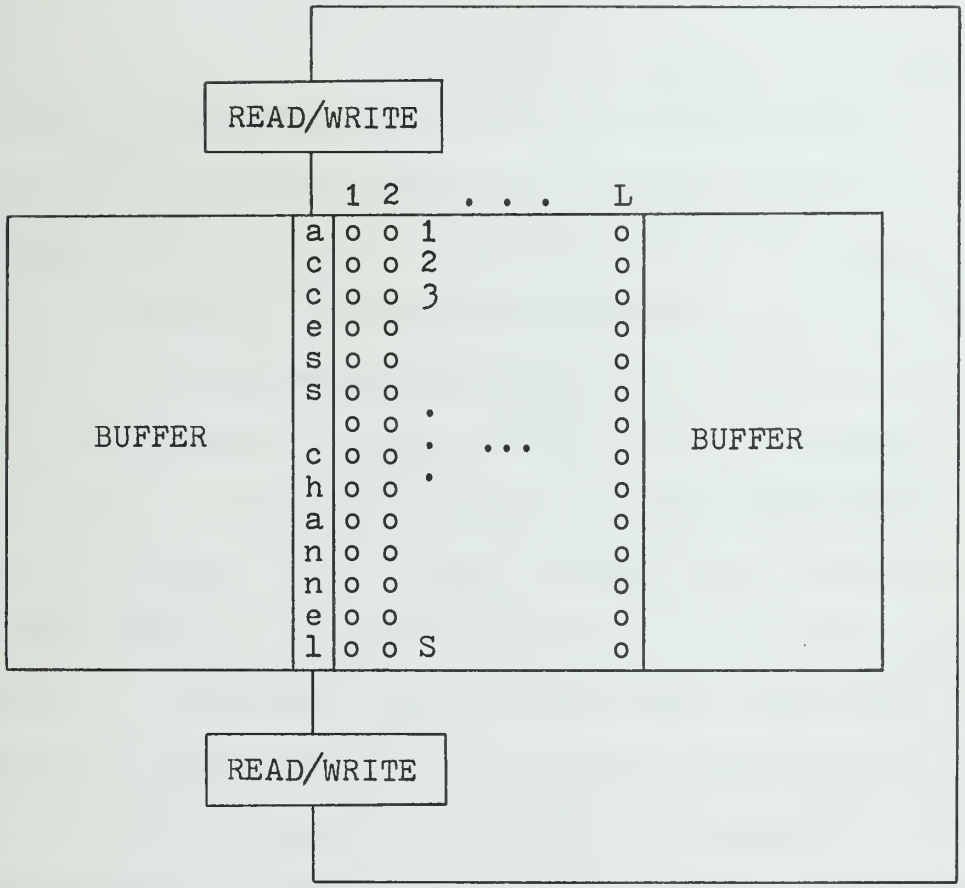


Figure 4. Bubble lattice organization.

dently controllable set of bubble memory chips. The number of bubble chips in a memory module, m, is a design parameter and it depends not only on the particular application but also on hardware constraints of magnetic bubble memory chips.

2.2. Data Rearrangement Operations

Three models of basic data rearrangement operations in magnetic bubble memories are described here. In all three models, data items or records are stored in a loop composed of N W-bit cells or locations, designated by integers 0,1,...,N-1. Location 0 in the loop is the only access port, i.e., an item must be brought to this location before it can be read out.

The first model, as shown in Figure 5, can be implemented in the uniform bubble ladder [20,21]. The flow steering switches S and T are controlled externally. In this case, N must be odd. Four basic data rearrangement operations, shown in Figure 6, are allowed. They are the global shift, the detached shift, the exchange and the delta exchange. We refer to these operations as operations (a), (b), (c) and (d), respectively. Each operation is assumed to take the time for W bubble shifts. The key field of a record may consist of any of 1 to W bit positions within the record. Comparison of key fields of records stored in adjacent cells is done as shown in Figure 7, bit serially.

The second model [16,17], shown in Figure 8, allows only two basic operations. The global shift, identical to

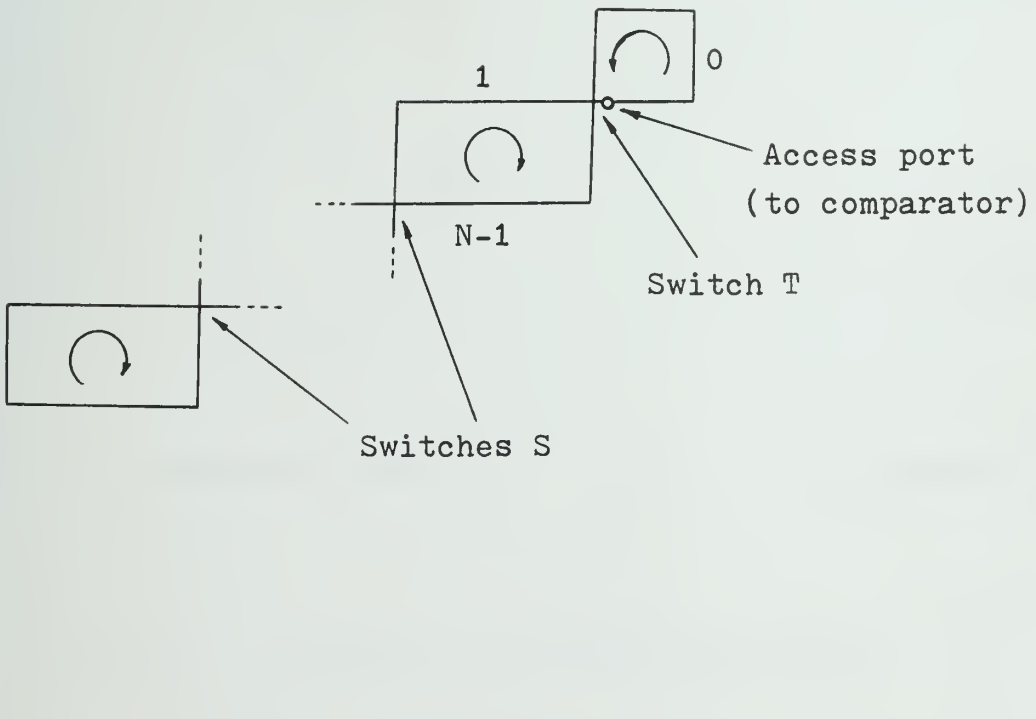
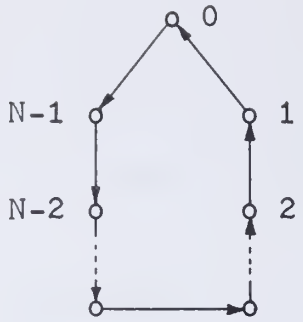
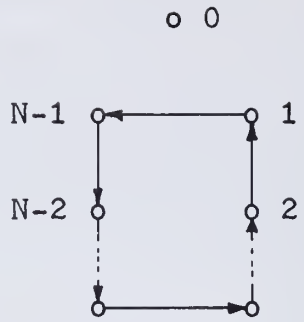


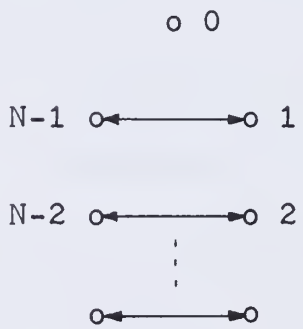
Figure 5. The uniform bubble ladder.



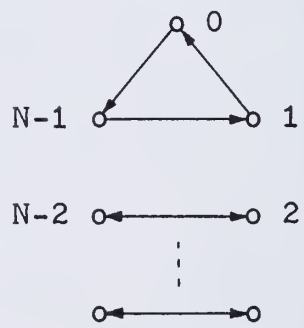
(a) Global shift



(b) Detached shift



(c) Exchange



(d) Delta exchange

Figure 6. The first model.

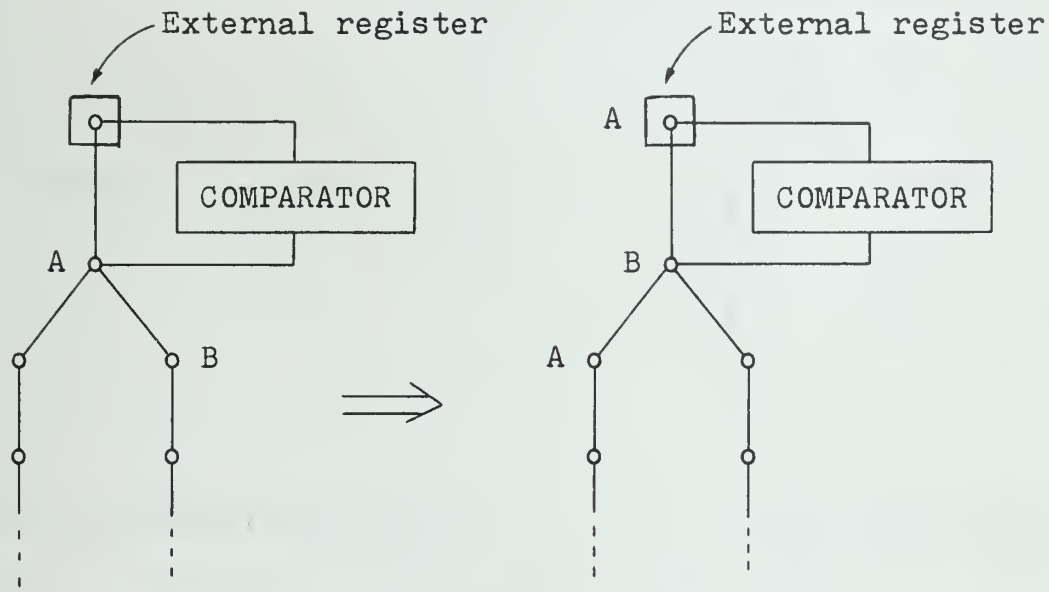
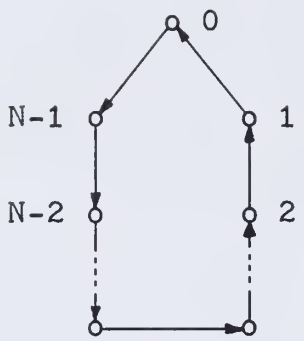
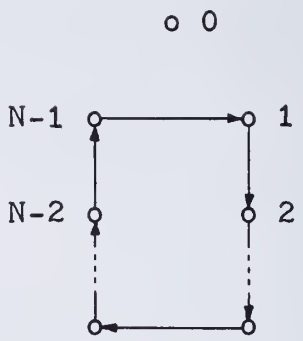


Figure 7. Comparison of items in data loop.



(a) Global shift



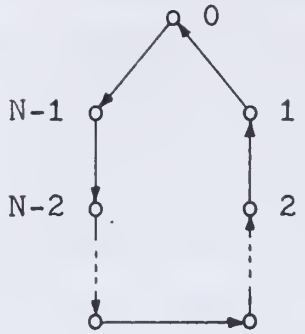
(e) Reversed detached shift

Figure 8. The second model.

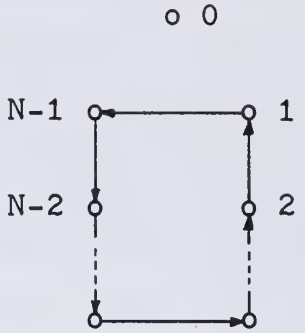
operation (a), and the reversed detached shift, referred to as operation (e). Each operation requires the time for one bubble shift. No restriction is placed on the parity of the number of cells. There are two ways to implement this model in the major-minor loop bubble chip organization [17]. It can be implemented in the major loops of W N-minor loop chips. In this case, each bit of a record is stored in a chip different from those containing all other bits of the record. Hence, all W bits of a record can be read out and compared externally in parallel, similar to the way shown in Figure 7. Alternatively, the model can be implemented in the N W-bit minor loops of a chip. In this case, a record is read out and compared bit serially, which requires the time for W bubble shifts.

The third model, shown in Figure 9, is being proposed here and its feasibility has not been proved. Similarly, four basic operations are allowed. Three of them are exactly the operations (a), (b) and (e), and the fourth operation, referred to as operation (f), is the reversed global shift. This model could be implemented in the bubble ladder, with the appropriate control of the driving magnetic field and of the switches S and T. In this case, all operations would require the time for W bubble shifts and the number of cells would have to be odd. If it were possible to implement in the major-minor loop organization, all operations would take one bubble shift and the number of cells could be even or odd, as in the second model.

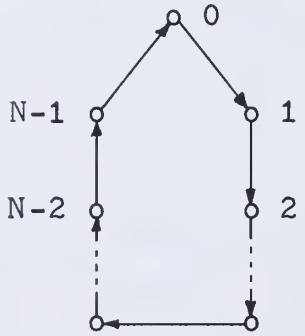
For the second and third models, unless explicitly



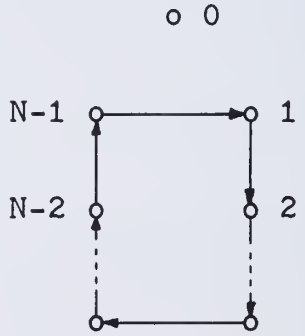
(a) Global shift



(b) Detached shift



(f) Reversed global shift



(e) Reversed detached shift

Figure 9. The third model.

indicated, we assume that data rearrangement operations take one bubble shift each and the number of cells can be even or odd.

2.3. Relational Data Model Concepts

In the relational data model, introduced by Codd [27,28,29], users of a data base management system view data as a set of tables, called relations. Each row in a relation is called a tuple and it represents an instance of the concept or entity described by the relation. Each relation has a column (or attribute) or set of columns (attributes), called the primary key, whose values must be unique in the tuples. Therefore, tuples of a relation are distinct (no duplicate tuples are allowed). The ordering of tuples is immaterial. The number of tuples in a relation is the cardinality of the relation.

The number of attributes in a relation is fixed and it is called the degree of the relation. The pool of values from which the actual values appearing in a given attribute are drawn is called a domain. A domain constitutes a well defined set of values and it is named. In a normalized relation, domains are simple (a value in a domain can not be another domain or relation). More than one attribute in the same relation can be based upon the same domain. To avoid confusion among attributes drawn from the same domains, it is customary to give role names to attributes.

As examples, four normalized relations, EMPLOYEE, EMPLOYEESKILL, PROJECT and JOBDESCRIPTION, are shown in

Figure 10. EMPLOYEE is a relation of degree 3 and cardinality 6, with attributes EMP (employee), PROJ (project) and SALARY (salary), and its primary key is EMP. PROJECT has attributes PROJ, COMPLDATE (completion date) and LOC (location). The primary keys of PROJECT, EMPLOYEESKILL and JOBDESCRIPTION are PROJ, (EMP,SKILL) and (JOBNAME,SKILL), respectively.

Relational algebra is a collection of high-level operations on relations, which operate on entire relations and yield relations as results of their application. There are two groups of relational algebraic operations, the traditional set operations (union, intersection, difference and Cartesian product) and the special relational operations (selection, projection, join and division).

The result of the selection operation, expressed as Selection(Relation,Condition), is the relation composed of the tuples from Relation which satisfy Condition. Relations R1 and R2 in Figure 11 are the results of selection on JOBDESCRIPTION and PROJECT, respectively.

The result of the projection operation, expressed as Projection(Relation,Domainset) (where Domainset is a subset of the domains on which Relation is defined), is the relation obtained by eliminating from Relation the columns not in Domainset (eventual duplicate tuples must be discarded). Relation R5 in Figure 11 is the result of a projection on R3.

The result of the join operation, expressed as Join(Relation1,Domain1,Condition,Relation2,Domain2), is the

EMPLOYEE

EMP	PROJ	SALARY
E1	P1	20000
E2	P1	10000
E3	P2	15000
E4	P3	13000
E5	P4	25000
E6	P2	20000

JOBDESCRIPTION

JOBNAME	SKILL
J1	S1
J1	S2
J2	S3
J3	S4
J4	S5
J5	S3
J6	S4

EMPLOYEESKILL

EMP	SKILL
E1	S1
E1	S2
E2	S3
E3	S4
E4	S5
E5	S3
E5	S4
E6	S1
E6	S2

PROJECT

PROJ	COMPLDATE	LOC
P1	JUN	L1
P2	MAR	L2
P3	DEC	L2
P4	MAR	L1

Figure 10. Examples of relations.

R1 = Selection(JOBDESCRIPTION,JOBNAME = "J1")

R1

JOBNAME	SKILL
J1	S1
J1	S2

R2 = Selection(PROJECT,COMPLDATE "MAR")

R2

PROJ	COMPLDATE	LOC
P2	MAR	L2
P4	MAR	L1

R3 = Join(EMPLOYEE,PROJ,"=",R2,PROJ)

R3

EMP	PROJ	SALARY	PROJ	COMPLDATE	LOC
E3	P2	15000	P2	MAR	L2
E5	P4	25000	P4	MAR	L1
E6	P2	20000	P2	MAR	L2

R4 = Division(EMPLOYEESKILL,SKILL,SKILL,R1)

R4

EMP
E1
E6

R5 = Projection(R3,LOC)

R5

LOC
L2
L1

Figure 11. Examples of relational algebraic operations.

relation composed of tuples formed by the concatenation of tuples from Relation1 and tuples from Relation2 whose values in Domain1 and Domain2, respectively, satisfy Condition. Relation R3 in Figure 11 is the result of a join involving relations R2 and EMPLOYEE.

The division operation can be expressed as Division(Relation1,Domainset1,Domainset2,Relation2). A tuple t from Projection(Relation1,domains not in Domainset1) is in the result relation if all tuples in Projection(Relation2, Domainset2) occur in tuples from Relation1 which are identical to t in the domains of Relation1 not in Domainset1. Relation R4 in Figure 11 is the result of a division involving relations EMPLOYEESKILL and R1.

In the two examples below we show how relational algebraic operations can be used to get the answer to queries to the data base.

Example 1: Get all employees capable of doing job J1 and who will be free after March.

R1 = Selection(JOBDESCRIPTION,JOBNAME = "J1")

R4 = Division(EMPLOYEESKILL,SKILL,SKILL,R1)

R2 = Selection(PROJECT,COMPLDATE = "MAR")

R3 = Join(EMPLOYEE,PROJ,"=",R2,PROJ)

R6 = Join(R4,EMP,"=",R3,EMP)

Result = Projection(R6,EMP) =

EMP
E6

Example 2: List all skills of employees currently working in project P2.

R7 = Selection(EMPLOYEE,PROJ = "P2")

R8 = Join(R1,EMP,"=",EMPLOYEESKILL,EMP)

Result = Projection(R2,SKILL) =

SKILL
S4
S1
S2

In Chapter 5, algorithms for the relational operations described above (plus the traditional set operations) will be discussed.

3. MEMORY ORGANIZATION

Efficient access of data and optimal utilization of memory space are the subjects of major concern in the design of memory organizations in data base management systems. An organization that allows efficient access of data may be wasteful as far as space utilization is concerned. An organization that makes efficient use of space may result in complex control and time consuming data access mechanisms. Furthermore, most storage media are available in modules of fixed size, giving rise to problems of fragmentation (unused space) and space overflow (data in excess of module capacity). Usually, a designer needs to find a compromise among his objectives.

In this chapter, we are concerned with the evaluation of several bubble memory organizations. In particular, we evaluate retrieval times of data for different magnetic bubble memory organizations. Results obtained are then used to evaluate memory utilization in hierarchical memory systems. Our discussions are based on the three basic bubble chip organizations described in Section 2.1.

3.1. Bubble Memory Organization

For the purpose of evaluating alternatives of bubble memory organization, we consider a memory composed of C chips. The memory is organized as P pages with N W-bit words each. Therefore, the capacity of the memory is CB = PNW bits.

There are three possible ways of distributing words and bits of a page among chips. All words of a given page can be stored in the same chip - page per chip organizations. A chip may contain many pages. That is, B/NW may be larger than one. All bits in a word can be stored in the same storage loop - page per chip/word per loop organization ($W \leq S$). Alternatively, bits in a word may be distributed among storage loops in a chip - page per chip/word distributed organization ($W \leq L$). In page per chip organizations, concurrent access to C different pages can be achieved by the provision of the necessary buffer space for retrieved words and independent control for each chip in the memory. Words are accessed in a bit serial mode.

Another way is to have N words in a page stored in N different chips (and hence one word from each of B/N pages are stored in the same chip) - word per chip organizations. Depending on the way bits in a word are stored in storage loops, we also have two alternatives, word per chip/word per loop ($W \leq S$) or word per chip/word distributed ($W \leq L$). Again, words are accessed bit serially and concurrent access can be achieved by the provision of buffer space to store retrieved words and independent control for each chip in the memory. Up to C words, no more than one of them from the same chip,could be retrieved concurrently. A single control for the whole memory would suffice for the parallel retrieval of all words from C/N pages.

Finally, we can have one bit from each word stored in a chip. In other words, W chips are used to store one word.

Therefore, words are accessed in a bit parallel mode. This is the most constrained of the three general organizations considered. A similar organization, using the major-minor loop chip organization, is proposed by Hsiao and Kannan [10], for their Bucket Memory System. There are two ways to store the N words of a page. When bits in the N words of a page are stored in the same W chips, we have bit per chip/page per loop ($N \leq S$) and bit per chip/page distributed ($N \leq L$) organizations. One bit from each word of B/N pages are stored in the same chip. In this case, words in a page can be accessed only sequentially - bit per chip sequential organizations. The other alternative is to use NW chips to store all bits in a page and, hence, making the words in the page at the access ports randomly accessable - bit per chip random memory organizations. In this case, one bit from each of B pages are stored in the same chip.

The bubble memory organizations described above are shown schematically in Figures 12 - 15. Table 1 summarizes the relationships among the different parameters for these memory organizations. We assume that the values of the logical parameters of the memory, W and N, fit the values of the physical parameters of the bubble memory chips, B, L, and S. That is, B/NW, L/NW, NW/L, S/NW, NW/S, L/W, L/N, S/W and S/N are considered to be integers where it is convenient to do so to simplify the evaluation of retrieval times.

The choice of any particular memory organization in a given system is often based on considerations other than re-

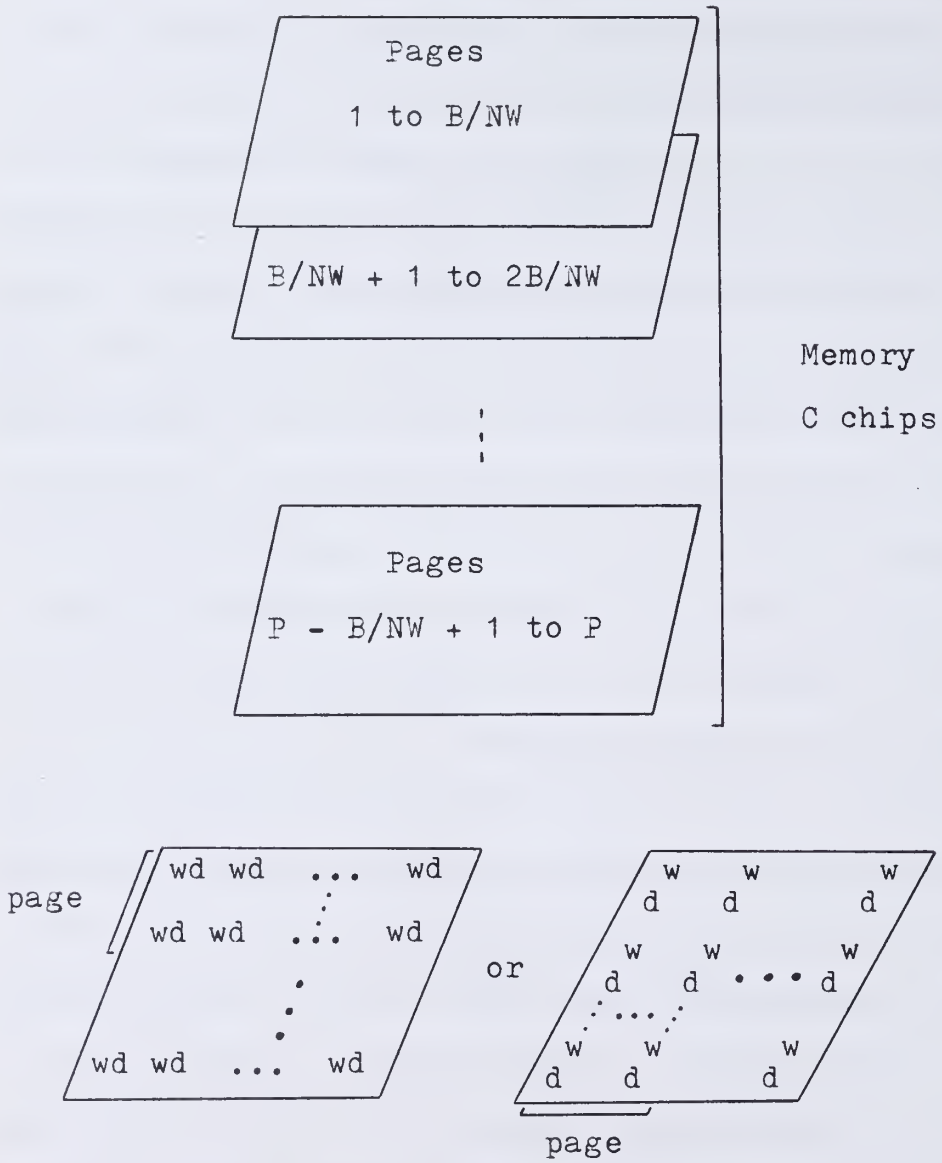


Figure 12. Page per chip memory organizations.

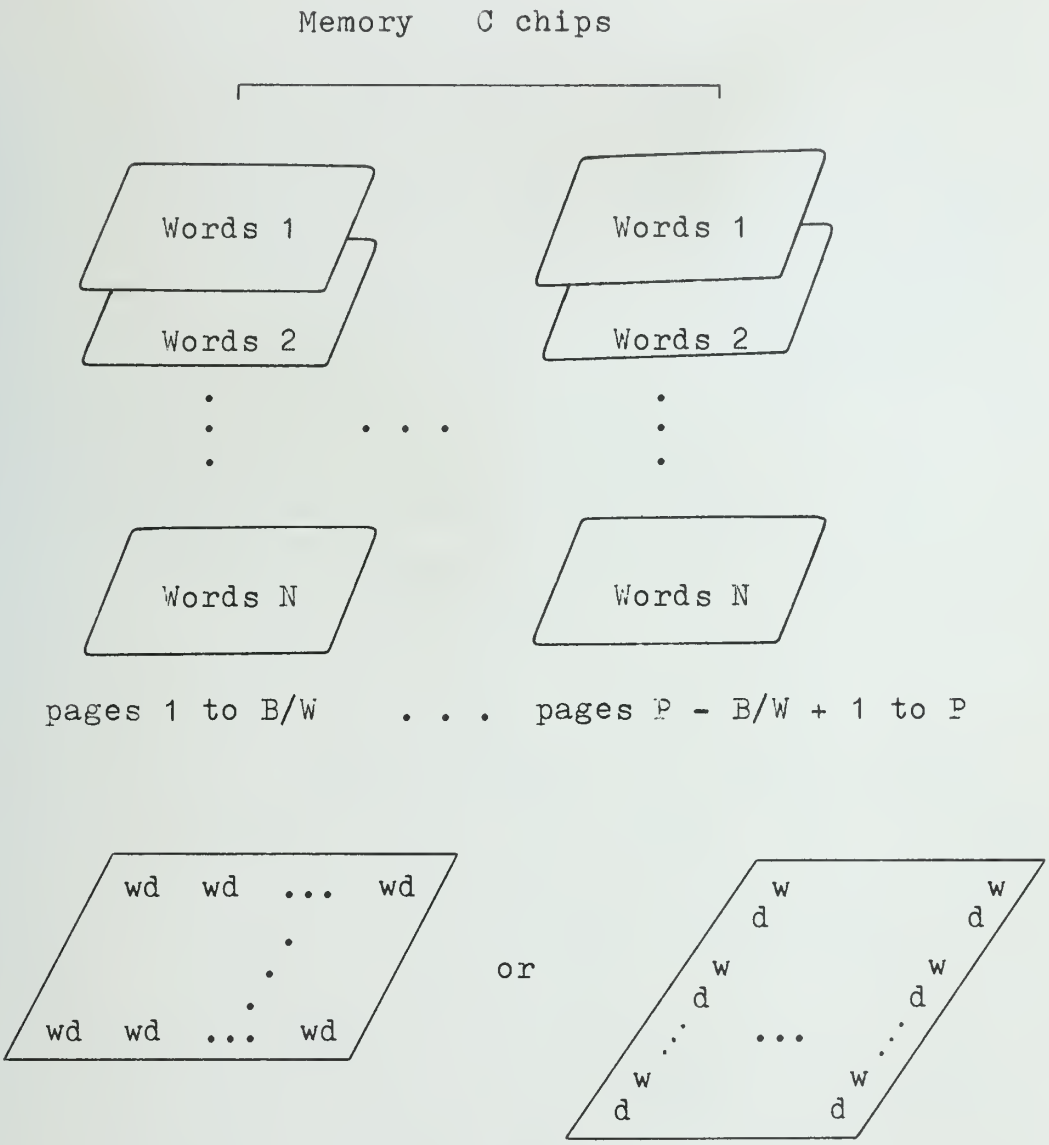


Figure 13. Word per chip memory organizations.

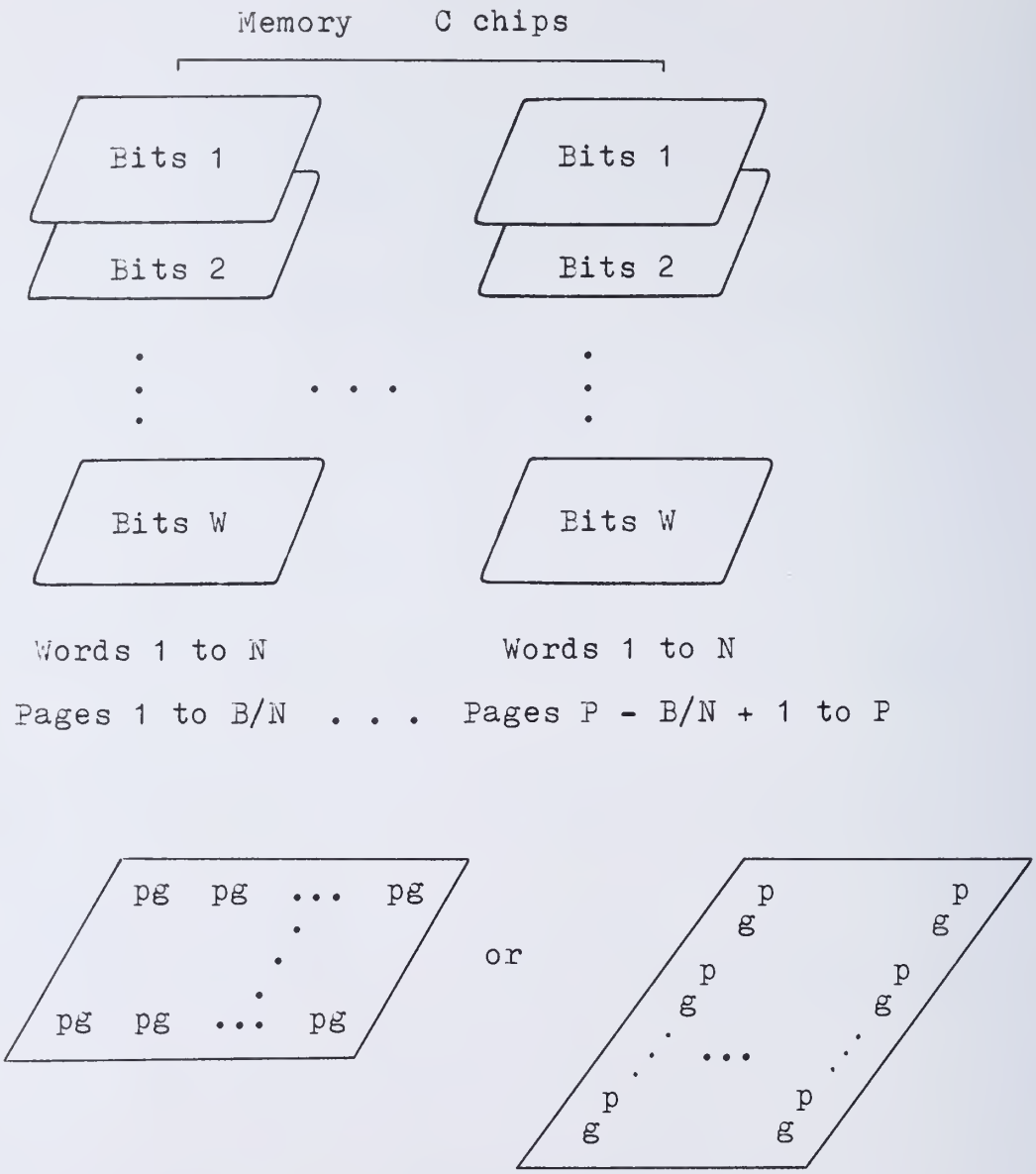


Figure 14. Bit per chip sequential memory organizations.

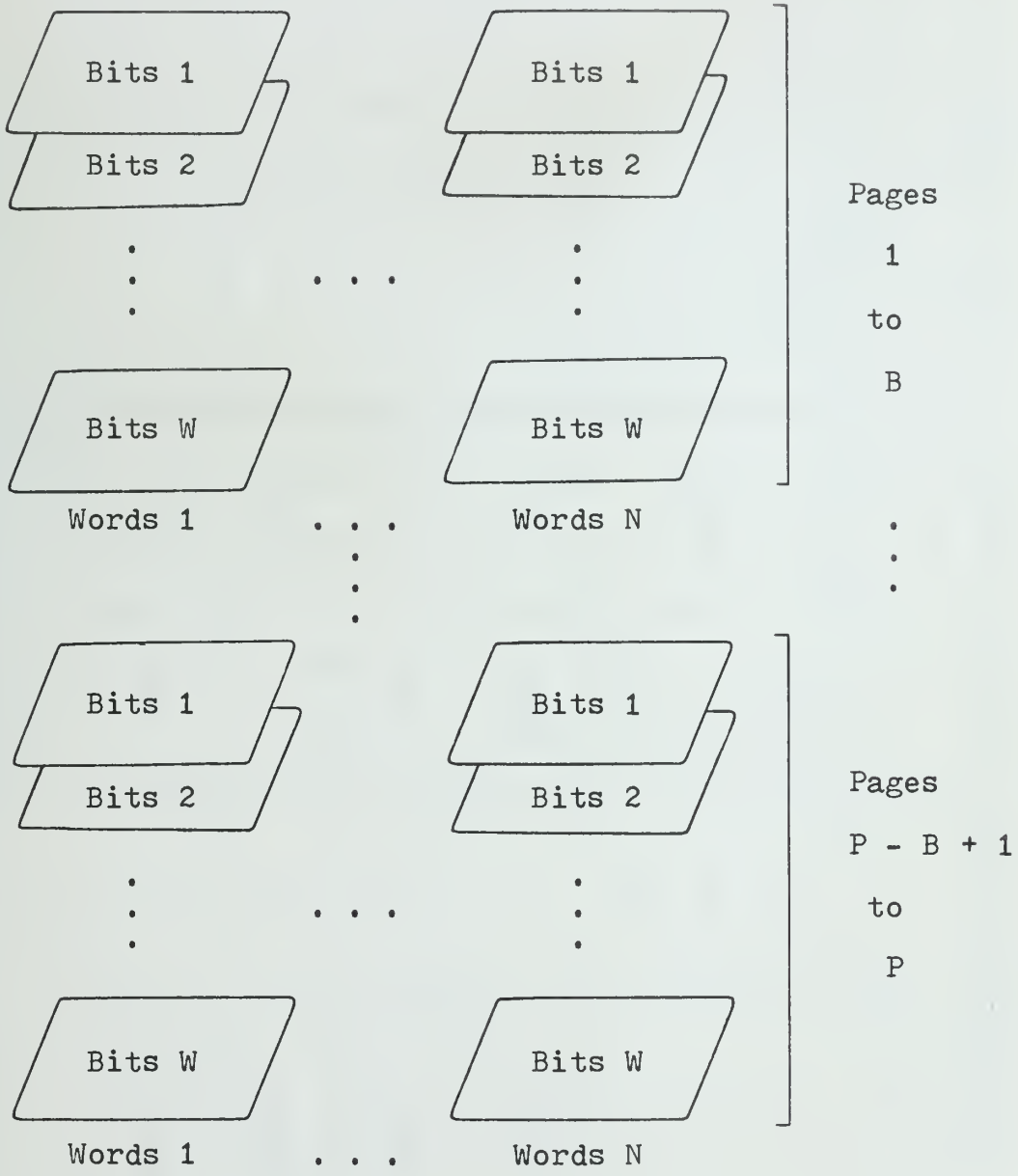


Figure 15. Bit per chip random memory organizations.

Table 1. Relationships among Different Parameters in Memory Organizations

		No. of Chips per Page	No. of Pages on a Chip	No. of Concurrently Accessable		
				Pages	Words per Page	Bits per Word
Page per Chip		1	B/NW	C	1	1
Word per Chip		N	B/W	C/N	N	1
Bit per Chip	Random	NW	B	C/NW	N	W
	Sequential	W	B/N	C/W	1	W

trieval times. For example, in applications where it is essential to safeguard the correctness of individual memory words in case of catastrophic chip failures, bit per chip organization is often selected. By using redundant chips and error correction codes (with W information bits), it is relatively simple to assure correctness of individual words in bit per chip organization. This error control scheme, if implemented for the two other organizations would require more complex control (and possibly more buffer space). (For example, in the case of word per chip organization, error correction can be done for the N bits (one bit per word in the page) at the access ports of the N chips containing the page.) On the other hand, as mentioned above, more words in a page or more pages can be accessed in parallel in the case of word and page per chip organizations.

3.2. Average Retrieval Times

We let T_w and T_p denote the average retrieval times per word and per page, respectively. T_w and T_p are expressed in terms of the numbers of shift operations required to move all bits in a randomly selected word and page to the access port per bit retrieved.

To evaluate these two quantities, we assume that the direction of shift yielding the least number of shifts required for access of a word (or page) is always chosen. All words and pages have the same probability of being selected. For example, in a loop of length S, the number of shifts needed to bring a

bit from position K to the first position in the loop is K-1 mod ($\lfloor S/2 \rfloor$ +1), for K $\leq \lfloor S/2 \rfloor$ +1, and (S-K+1) mod ($\lfloor S/2 \rfloor$ +1), otherwise. Furthermore, the expected number of shifts to bring any bit to the first position in the loop is S/4 shifts, if S is even, and $(S^2-1)/4S$ (approximately S/4) shifts, if S is odd.

Expressions for average retrieval times, for the three basic bubble chip organizations of Section 2.1 and the memory organizations described in the previous section, are shown in Tables 2 - 5. These tables do not include all the possible alternatives of bubble memory organization. When variations of alternatives are possible, the choice was based on simplicity of the control mechanism and/or retrieval efficiency. Expressions are given for both cases of exclusion and inclusion (when applicable) of the number of shifts required, after retrieval of data, to move data back to their original positions. For the on-chip decoder organization, expressions are also given for the case of stepped control of address lines (described in Section 2.1). Details on the derivation of the expressions for retrieval times are presented in Appendix A.

A simple inspection of Tables 2 - 5 reveals that the bit per chip random organization yields the best average page retrieval times for all bubble chip organizations. (In particular, the best average page retrieval time is achieved with the on-chip decoder organization.) An expected result is that average word retrieval times are the same for the corresponding

Table 2. Average Retrieval Times in Page per Chip Memory Organizations

2.1. Major-minor Loop Chip Organization

Word per Loop

	$S = W$	$S > W$
T_w	$4 + B/W$ (*)	$4 + B/W + S/2W$ (*)
	$2 + B/2W$	$2 + B/2W + S/4W$

	$S \leq NW$	$S > NW$
T_p	$4 + B/NW$ (*)	$4 + B/NW + S/2NW$ (*)
	$2 + B/2NW$	$2 + B/2NW + S/4NW$

Word Distributed

	$B/S = W$	$B/S > W$
T_w	$2 + 2/W + S/2W$ (*)	$2 + 2/W + B/2SW + S/2W$ (*)
	$1 + 1/W + S/4W$	$1 + 1/W + B/4SW + S/4W$

	$B/S \leq NW$	$B/S > NW$
T_p	$2 + 2S/B + S/2NW$ (*)	$2 + 2/NW + B/2SNW + S/2NW$ (*)
	$1 + S/B + S/4NW$	$1 + 1/NW + B/4SNW + S/4NW$

(*) Including shifts to move data back to original positions.

Table 2. Average Retrieval Times in Page per Chip Memory Organizations (continued)

2.2. On-chip Decoder Chip Organization

Word per Loop

	$S = W$	$S > W$
T_w	$1 + 2a/W$ (*)	$2 + 2a/W + S/2W$ (*)
	$1 + a/W$	$1 + a/W + S/4W$

	$S \leq NW$	$S > NW$
T_p	$1 + 2a/S$ (*)	$2 + 2a/NW + S/2NW$ (*)
	$1 + a/S$	$1 + a/NW + S/4NW$
	$1 + 2a/NW$ (*) (+)	
	$1 + a/NW$ (+)	

Word Distributed

T_w	$2 + 2a + S/2W$ (*)	$2 + 2a/W + S/2W$ (*) (+)
	$1 + a + S/4W$	$1 + a/W + S/4W$ (+)
T_p	$2 + 2a + S/2NW$ (*)	$2 + 2a/NW + S/2NW$ (*) (+)
	$1 + a + S/4NW$	$1 + a/NW + S/4NW$ (+)

$a = \log_2 B/S$

(*) Including shifts to move data back to original positions.

(+) Using stepped control of address lines.

Table 2. Average Retrieval Times in Page per Chip Memory Organizations (continued)

2.3. Bubble Lattice Chip Organization

Word per Column

	$S = W$	$S > W$
T_w	$1 + 1/W + B/SW$ (*)	$2 + 1/W + B/SW + S/2W$ (*)
	$1 + 1/2W + B/2SW$	$1 + 1/2W + B/2SW + S/4W$

	$S \leq NW$	$S > NW$
T_p	$B/SNW + (S+S^2)/(NW)^2$ (*)	$2 + 1/NW + B/SNW + S/2NW$ (*)
	$B/2SNW + (S/2+S^2)/(NW)^2$	$1 + 1/2NW + B/2SNW + S/4NW$

Word Distributed

	$B/S = W$	$B/S > W$
T_w	$4 + S/2$ (*)	$3 + B/SW + S/2$ (*)
	$2 + S/4$	$3/2 + B/2SW + S/4$

	$B/S \leq NW$	$B/S > NW$
T_p	$4 + S/2$ (*)	$3 + B/SNW + S/2$ (*)
	$2 + S/4$	$3/2 + B/2SNW + S/4$

(*) Including shifts to move data back to original positions.

Table 3. Average Retrieval Times in Word per Chip Memory Organizations

3.1. Major-minor Loop Chip Organization

Word per Loop

	S = W	S > W
T_w	4 + B/W (*)	4 + B/W + S/2W (*)
	2 + B/2W	2 + B/2W + S/4W
	S = W	**S > W**
T_p	4/N + B/NW (*)	4/N + B/NW + S/2NW (*)
	2/N + B/2NW	2/N + B/2NW + S/4NW

Word Distributed

	B/S = W	B/S > W
T_w	2 + 2/W + S/2W (*)	2 + 2/W + B/2SW + S/2W (*)
	1 + 1/W + S/4W	1 + 1/W + B/4SW + S/4W
	B/S = W	**B/S > W**
T_p	2/N + 2/NW + S/2NW (*)	2/N + 2/NW + B/2SNW + S/2NW (*)
	1/N + 1/NW + S/4NW	1/N + 1/NW + B/4SNW + S/4NW

(*) Including shifts to move data back to original positions.

Table 3. Average Retrieval Times in Word per Chip Memory Organizations (continued)

3.2. On-chip Decoder Chip Organization

Word per Loop

	S = W	S > W
T_w	1 + 2a (*)	2 + 2a + S/2W (*)
	1 + a	1 + a + S/4W

	S = W	S > W
T_p	1/N + 2a/NW (*)	2/N + 2a/NW + S/2NW (*)
	1/N + a/NW	1/N + a/NW + S/4NW

Word Distributed

T_w	2 + 2a + S/2W (*)	2 + 2a/W + S/2W (*) (+)
	1 + a + S/4W	1 + a/W + S/4W (+)
T_p	2/N + 2a/N + S/2NW (*)	2/N + 2a/NW + S/2NW (*) (+)
	1/N + a/N + S/4NW	1/N + a/NW + S/4NW (+)

$a = \log_2 B/S$

(*) Including shifts to move data back to original positions.

(+) Using stepped control of address lines.

Table 3. Average Retrieval Times in Word per Chip Memory Organizations (continued)

3.3. Bubble Lattice Chip Organization

Word per Column

	S = W	S > W
T_w	1 + 1/W + B/SW (*)	2 + 1/W + B/SW + S/2W (*)
	1 + 1/2W + B/2SW	1 + 1/2W + B/2SW + S/4W

	S = W	S > W
T_p	1/N + 1/NW + B/SNW (*)	2/N + 1/NW + B/SNW + S/2NW (*)
	1/N + 1/2NW + B/2SNW	1/N + 1/2NW + B/2SNW + S/4NW

Word Distributed

	B/S = W	B/S > W
T_w	4 + S/2 (*)	3 + B/SW + S/2 (*)
	2 + S/4	3/2 + B/2SW + S/4

	B/S = W	B/S > W
T_p	4/N + S/2N (*)	3/N + B/SNW + S/2N (*)
	2/N + S/4N	3/2N + B/2SNW + S/4N

(*) Including shifts to move data back to original positions.

Table 4. Average Retrieval Times in Bit per Chip Sequential Memory Organizations

4.1. Major-minor Loop Chip Organization

Page per Loop

T_w	$4/W + B/2SW + S/2W$ (*)	$2/W + B/4SW + S/4W$
	$S \leq N$	$S > N$
T_p	$BS/N^2W + 4S^2/N^2W$ (*)	$4/W + B/NW + S/2NW$ (*)
	$BS/2N^2W + 2S^2/N^2W$	$2/W + B/2NW + S/4NW$

Page Distributed

T_w	$4/W + B/2SW + S/2W$ (*)	$2/W + B/4SW + S/4W$
	$B/S \leq N$	$B/S > N$
T_p	$2b/NW + B/2N^2W + 2b^2/W$ (*)	$2/W + 2/NW + b/2W + S/2NW$ (*)
	$b/NW + B/4N^2W + b^2/W$	$1/W + 1/NW + b/4W + S/4NW$

$b = B/SN$

(*) Including shifts to move data back to original positions.

Table 4. Average Retrieval Times in Bit per Chip Sequential Memory Organizations (continued)

4.2. On-chip Decoder Chip Organization

Page per Loop

T_w	$2/W + 2a/W + S/2W$ (*)	$1/W + a/W + S/4W$
	$S \leq N$	$S > N$
T_p	$S^2/N^2W + 2aS/N^2W$ (*)	$2/W + S/2NW + 2a/NW$ (*)
	$S^2/N^2W + aS/N^2W$	$1/W + S/4NW + a/NW$

Page Distributed

T_w	$2/W + 2a/W + S/2W$ (*)	$1/W + a/W + S/4W$
	$B/S \leq N$	$B/S > N$
T_p	$B/2N^2W + \frac{2b^2}{W}(1 + a)$ (*)	$S/2NW + \frac{2}{W}(1 + a)$ (*)
	$B/4N^2W + \frac{b^2}{W}(1 + a)$	$S/4NW + \frac{1}{W}(1 + a)$
	$B/2N^2W + \frac{2b^2}{W} + \frac{2ba}{NW}$ (*)(+)	$S/2NW + \frac{2}{W} + \frac{2a}{NW}$ (*) (+)
	$B/4N^2W + \frac{b^2}{W} + \frac{ba}{NW}$ (+)	$S/4NW + 1/W + a/NW$ (+)

$a = \log_2 B/S$; $b = B/SN$

(*) Including shifts to move data back to original positions.

(+) Using stepped control of address lines.

Table 4. Average Retrieval Times in Bit per Chip Sequential Memory Organizations (continued)

4.3. Bubble Lattice Chip Organization

Page per Column

T_w	$3/W + B/SW + S/2W$ (*)	$3/2W + B/2SW + S/4W$
	$S \leq N$	$S > N$
T_p	$B/N^2W+S/N^2W+S^2/N^2W$ (*)	$2/W + 1/NW + B/SNW + S/2NW$ (*)
	$B/2N^2W + S/2N^2W+S^2/N^2W$	$1/W + 1/2NW + B/2SNW + S/4NW$

Page Distributed

T_w	$3/W + B/SW + S/2W$ (*)	$3/2W + B/2SW + S/4W$
	$B/S \leq N$	$B/S > N$
T_p	$4B^2/S^2N^2W + B^2/2SN^2W$ (*)	$3/W + B/SNW + S/2W$ (*)
	$2B^2/S^2N^2W + B^2/4SN^2W$	$3/2W + B/2SNW + S/4W$

(*) Including shifts to move data back to original positions.

Table 5. Average Retrieval Times in Bit per Chip Random Memory Organizations

5.1. Major-minor Loop Chip Organization

T_w	$4/W + B/2SW + S/2W$ (*)	$2/W + B/4SW + S/4W$
T_p	$4/NW + B/2SNW + S/2NW$(*)	$2/NW + B/4SNW + S/4NW$

5.2. On-chip Decoder Chip Organization

T_w	$2/W + 2a/W + S/2W$ (*)	$1/W + a/W + S/4W$
T_p	$2/NW + 2a/NW + S/2NW$ (*)	$1/NW + a/NW + S/4NW$

5.3. Bubble Lattice Chip Organization

T_w	$3/W + B/SW + S/2W$ (*)	$3/2W + B/2SW + S/4W$
T_p	$3/NW + B/SNW + S/2NW$ (*)	$3/2NW + B/2SNW + S/4NW$

$a = \log_2 B/S$

(*) Including shifts to move data back to original positions.

bubble chip organizations and word distribution (word per loop and word distributed) in page per chip and word per chip organizations. This same result is true for the corresponding bubble chip organizations and all page distributions (page per loop or page distributed) in bit per chip organizations (sequential or random). Many of the alternative ways to arrange pages, words, or bits in a chip are obviously poor choices that have poor performance in terms of retrieval times and hence will not be considered in the discussion that follows. Furthermore, we will consider retrieval times without including the number of shifts required to bring data back to their original positions. In most of the organizations discussed, the inclusion of these shifts implies in doubling the number of shifts for data retrieval.

The values of average word retrieval times for the different memory and bubble chip organizations selected are plotted in Figures 16 and 17. As shown in these figures, on-chip decoder organization yields the best word retrieval times for all chip, loop and word sizes, and for all memory organizations. We must note, however, that the number of shifts required to retrieve a word being largest in bubble lattice organization does not imply in largest word retrieval time (in seconds) since the time required for each shift is smaller in this chip organization. (As shown in Figure 16, for $B \cong 10^6$ and $S = 1024$, the retrieval time (in seconds) in bubble lattice and on-chip decoder organizations becomes equal when shift operation in the former is 4-5 times faster. For chips with small storage loops,

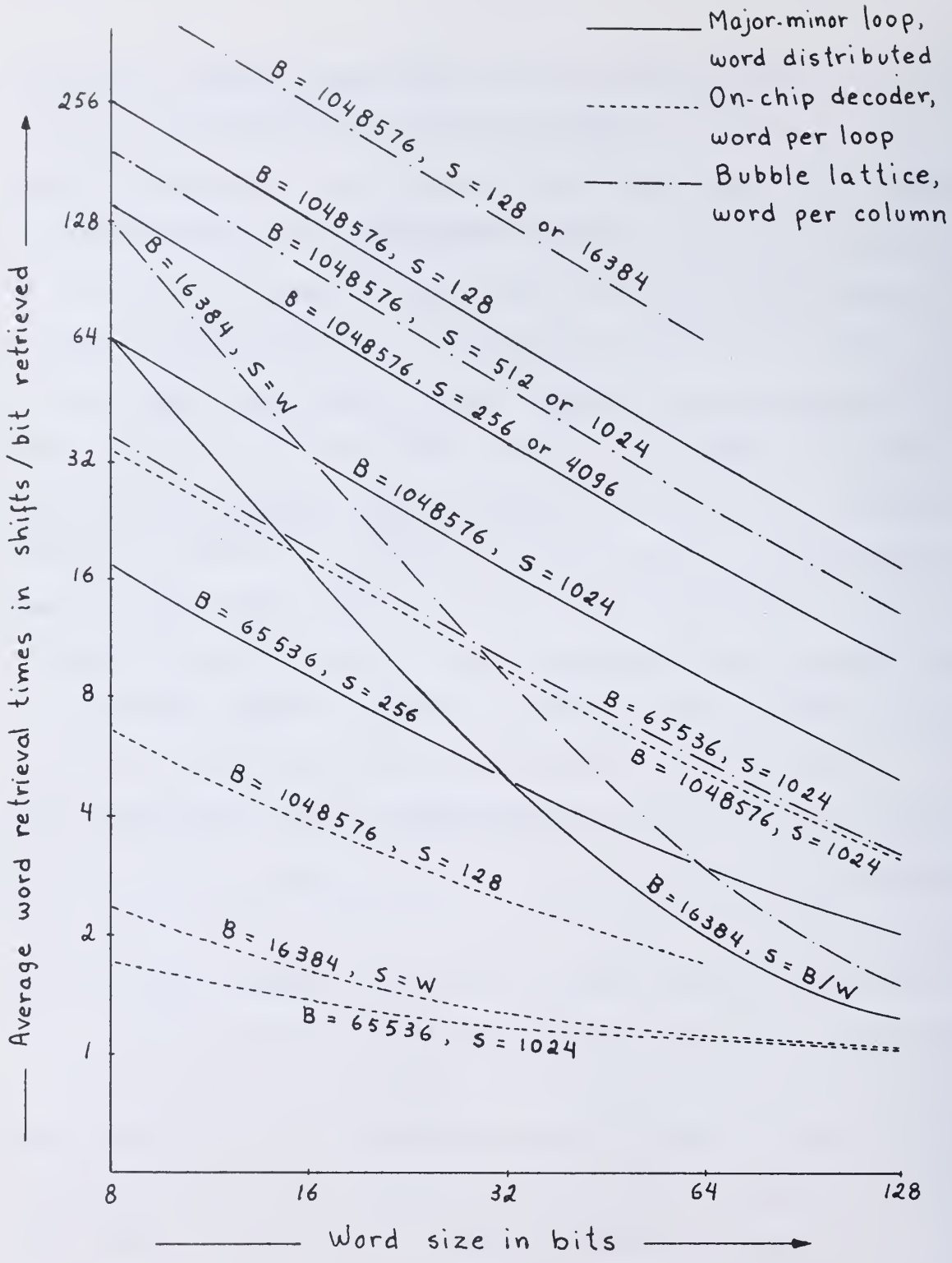


Figure 16. Average word retrieval times for page per chip and word per chip memory organizations.

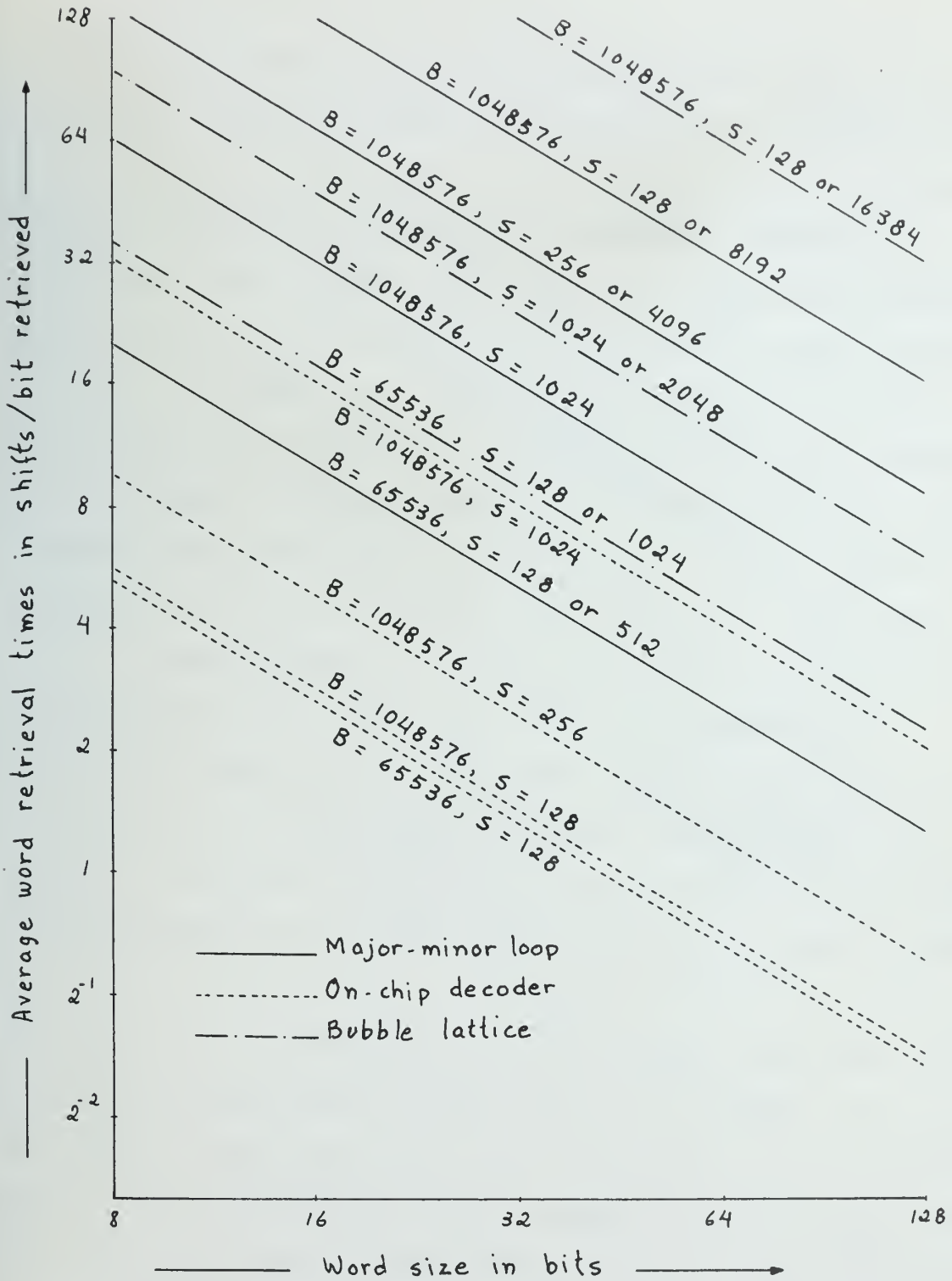


Figure 17. Average word retrieval times for bit per chip memory organizations.

e.g., S = 128, shift operations in bubble lattice need to be approximately 64 times faster to achieve the same retrieval times.)

A careful comparison of the plots in Figures 16 and 17 shows that for large chip sizes, the average word retrieval times are almost identical for the three memory organizations. This result is an expected one since the dominant term in average word retrieval times, in this case, is equal to the number of shifts required to bring a randomly selected bit to the access port. For small chip sizes and large word sizes, on the other hand, word retrieval times in bit per chip organization are smaller than in the other two organizations.

When compared in terms of average page retrieval times (See Figure 18), relative performance of bit per chip and word per chip organizations depends on the values of W and N. For the same chip sizes and chip organization, average retrieval time is smaller in word per chip organization whenever $N > W$.

3.3. Retrieval Times in Memory Hierarchies

We consider two bubble memory organizations, word per chip (shown in Figure 13) and bit per chip random (shown in Figure 15) for a hierarchical memory composed of C B-bit chips with major-minor loop chip organization. Each chip has $\sqrt{B}$ $\sqrt{B}$-bit minor loops. In the word per chip organization, the C chips are grouped into C/N modules. The N words in any page are stored in different chips of a module. Hence, the N words

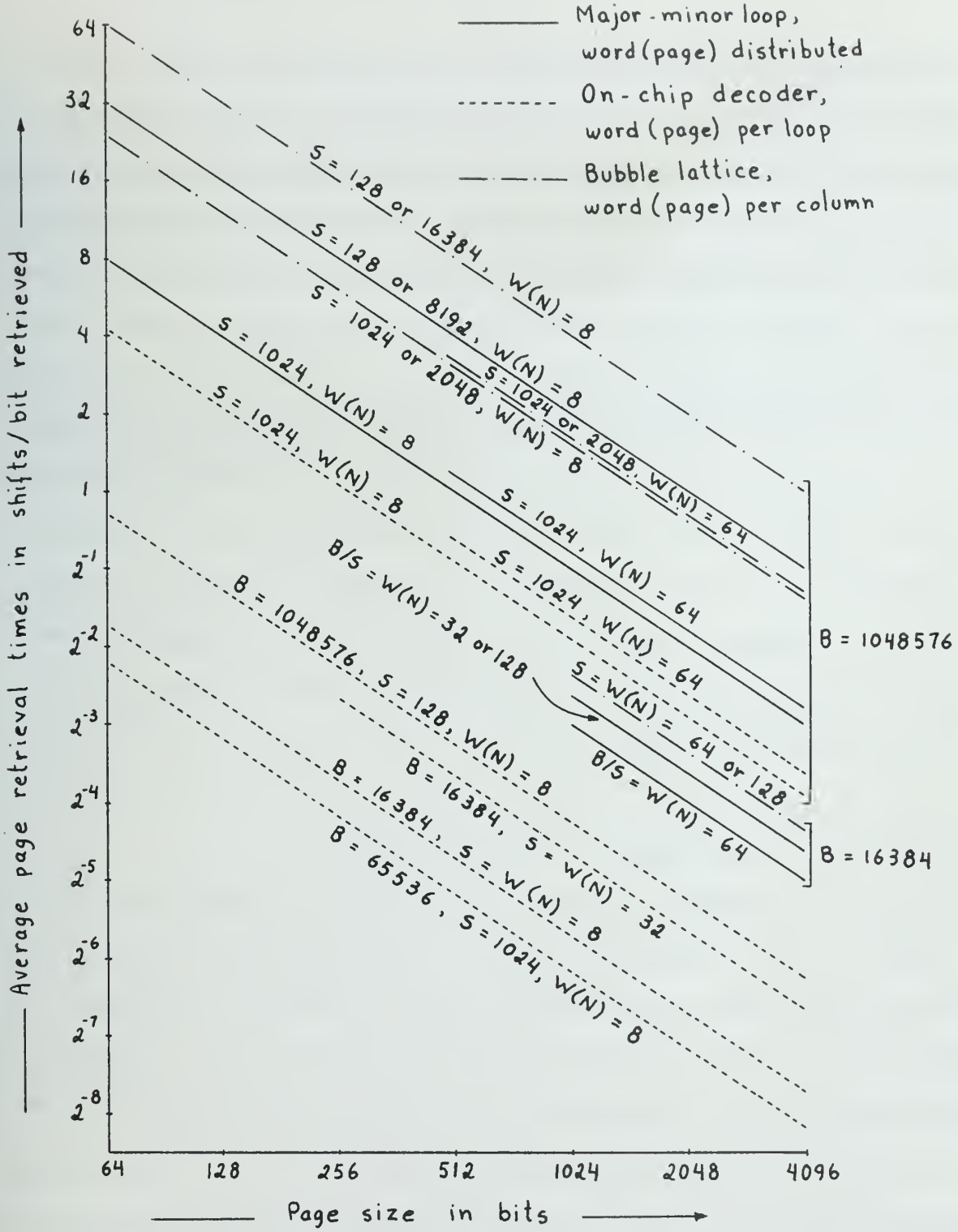


Figure 18. Average page retrieval times for word (and bit sequential) per chip memory organizations.

in the page at the access ports (read/write positions) can be randomly accessed and read out bit serially. These NW bit positions in all modules form a random access main memory containing C/N pages. We may view the rest of the memory as the secondary memory. This secondary memory may be further divided into two levels, with the second level memory consisting of those bit positions in the major loops not at the access ports. Similarly, in the bit per chip random organization, the chips are grouped into C/W modules. The N words in a page are stored in different modules. Words in the page at the access ports can be randomly accessed, with W bits available in parallel. However, the main memory, composed of these W bit positions in all modules, has a capacity of only C/NW bits. The secondary memory may also be divided into two levels.

3.3.1. Retrieval Times in Various Models

To compare the two organizations, we consider first a linear model of program behavior, due to Saltzer [30]. According to this model, the average number of memory references between page faults increases linearly with the size of the main memory, M. That is, the page fault rate, F_1, can be written as

$$F_1 = 1/aM, \quad M \geqslant 1 \qquad (3.1)$$

for some constant a. This crude model has been shown to describe reasonably well the combined paging behavior of a variable number of programs in dynamically multiprogrammed systems.

Let t_w and t_b be the average word retrieval times (number of shifts per bit retrieved) in word per chip and bit per chip random organizations, respectively. Assuming that pages not in the main memory are equally likely to be accessed, we have, for large chip sizes and page fault rate in Eq. (3.1),

$$t_w = 1 + \sqrt{B}/2aM_wW$$

where $M_w = C/N$ is the number of pages in main memory for word per chip organization. Similarly,

$$t_b = 1/W + \sqrt{B}/2aM_w$$

Hence, $t_b \geq t_w$ whenever $M_w \leq \sqrt{B}/2a$.

As an example, consider a two-level memory with a total capacity of 2^{30} (approximately 10^9) bits. When 2^{20}-bit chips are used, average word retrieval time in word per chip organization is smaller as long as $M_w \leq 512/a$. This inequality is valid for most practical choices of page sizes (a was observed to be in the order of 10 to 20). On the other hand, with smaller chip sizes (for example, if $\sqrt{B} = 256$, we have $t_w \leq t_b$ when $M_w = 8196/N \leq 128/a$), the inequality is valid only for very large page sizes.

Kuck and Lawrie [31] have observed from empirical data on program behavior in monoprogrammed systems that the mean time between page faults can be expressed as $\propto M^{\beta}$ for some constants $\propto$ and β. (For example, $\propto = 3.8$ and $\beta = 2.4$). In this case,

$$t_w = 1 + \sqrt{B}/2 \propto M_w^{\beta}W$$

and

$$t_b = 1/W + \sqrt{B}W^{\beta}/2 \propto M_w^{\beta}W$$

Hence, $t_w \leq t_b$ if

$$M_w \leq \sqrt{B}(W^{\beta}-1)/2 \propto (W-1) \qquad (3.2)$$

Baskett and Rafii [32] have observed that, for many different programs and replacement rules, page fault rate as a function of main memory size can be approximated as

$$F_2 = b \exp(-cM)$$

for constants $b\ (1 - 10^{-3})$ and $c\ (1 - up)$. Using this approximation, we have

$$t_w = 1 + (b\sqrt{B}/2W) \exp(-cM_w)$$

and

$$t_b = 1/W + (b\sqrt{B}/2W) \exp(-cM_w/W)$$

Again, $t_w \leq t_b$ if

$$2(W-1)/b\sqrt{B} \leq \exp(-cM_w/W) - \exp(-cM_w)$$

For most practical values of W ($W-1 \simeq W$), the second term in the right side of this inequality is negligible and it simplifies to

$$M_w \leq -\frac{W}{c} \log(2W/b\sqrt{B}) \qquad (3.3)$$

Finally, we consider the prefetching policy, SEPD(n), described in [33]. According to this policy, when a page fault occurs, n ($\leq$ C/NW or C/N) pages numbered consecutively from the page causing the page fault are brought into the main memory.

The n pages originally located at the access ports in the n modules are replaced. Let $m(x,n)$ denote the miss ratio for main memory size of x pages and n pages prefetched at page fault. Then,

$$t_w = 1 + (\sqrt{B}/2W)\ m(M_w,n)$$

and

$$t_b = 1/W + (\sqrt{B}/2W)\ m(M_w/W,n)$$

In this case, $t_w \leq t_b$ if

$$m(M_w,n) \leq m(M_w/W,n) - 2(W-1)/\sqrt{B} \qquad (3.4)$$

Average word retrieval times are smaller in word per chip organization when the inequality above is satisfied.

In Figure 19, we summarize the relative performance of the two organizations for the three program paging behavior models discussed above. The values plotted for N are the minimum page sizes for $t_w \leq t_b$. That is, in the regions of the graph above the threshold curves, the average word retrieval times in word per chip organization are smaller than those in bit per chip organization. As expected from inequalities (3.2) and (3.3) for the last two models of paging behavior, larger word sizes imply lower threshold curves.

3.4. Retrieval Times in Memories with Data Reordering

It has been shown that in a magnetic bubble memory using major-minor loop chip organization, one can reorder the data stored in memory in last use ordering. According to this

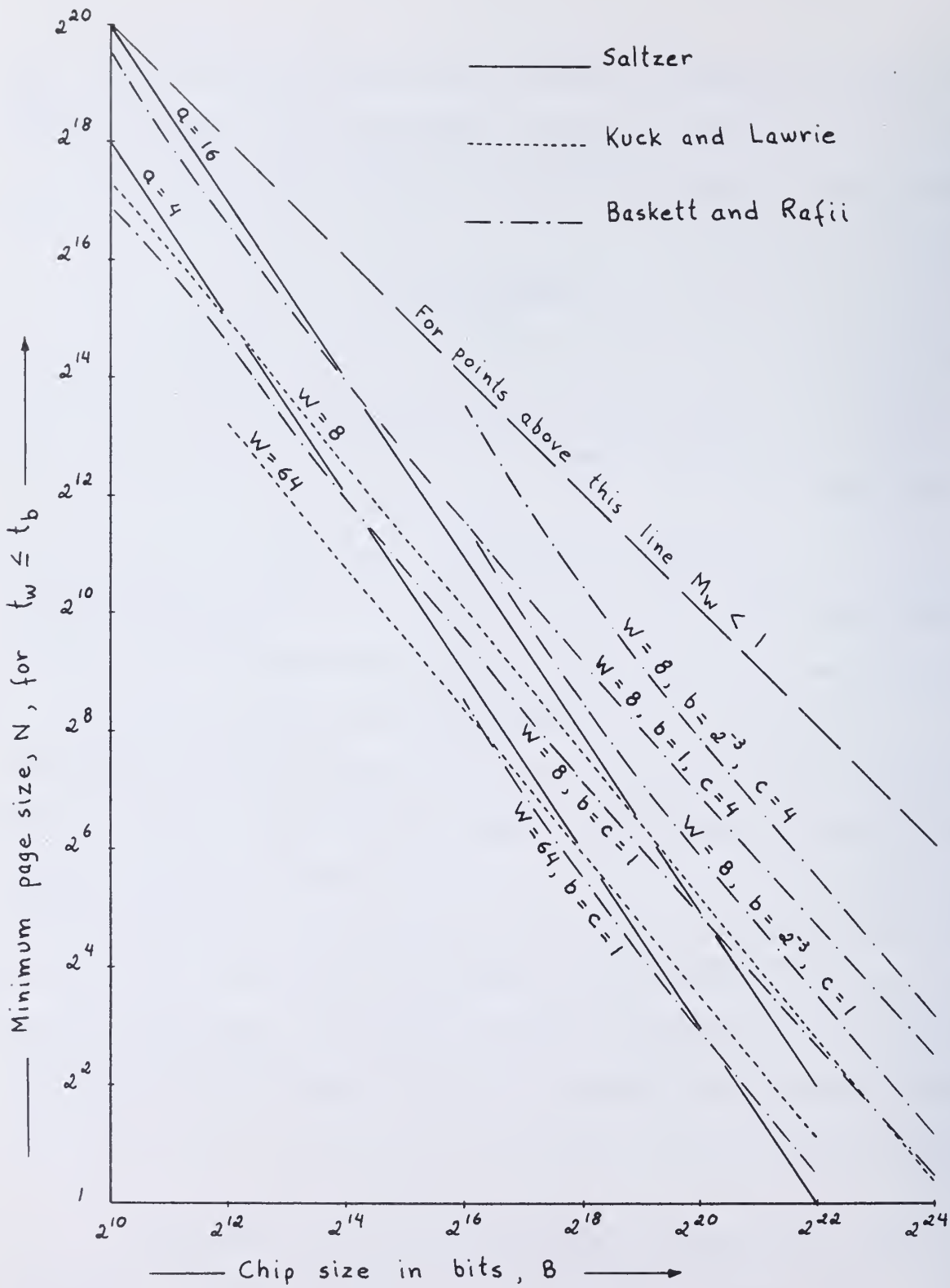


Figure 19. Relative performance of word per chip and bit per chip organizations for three models of program paging behavior (Total memory capacity of 2^{30} bits).

ordering, the more recently a page is used, the closer it is to the access port, and hence, a smaller number of shifts is required to access the page again [16,17]. In memories using bubble lattice chips and word per chip organization, one can dynamically reorder data in usage frequency ordering [18]. According to this ordering, the number of shift operations required to access a page is a decreasing function of the access frequency of that page.

4. ELEMENTARY FILE PROCESSING OPERATIONS

Sorting, merging and clustering of data items stored in magnetic bubble memories are discussed here. Algorithms for these data manipulation operations are designed and evaluated for the three models of basic data rearrangement operations described in Section 2.2.

4.1 Sorting Algorithms

Sorting of a sequence of elements is the process of rearranging the elements so that they appear in non-decreasing or non-increasing order, based on the values of a key associated with each element. The value of a key associated with an element identifies the element and determines its position in the ordered sequence. In our discussion, we consider that items stored in a loop of N cells are sorted when they are arranged in non-decreasing order of their key values, starting from location 0 in the loop.

In all sorting algorithms discussed here, we assume that the initial positions of the items stored inside an N-cell loop are not known. Keys are embedded in the items and are not stored elsewhere. Only one comparator is provided in each loop. Thus, comparison can be carried out only by bringing items to the access port (location 0). Figure 7 illustrates how two items, A and B, are positioned inside a loop during the comparison of their keys. Item A (in addition to being in an external register)

is one location past the access port and item B is at the access port. Furthermore, comparison should be made between keys of items in adjacent cells, in order to minimize data movement. Consequently, the sort algorithm which yields a minimal number of data rearrangement operations is the bubble sort [34].

In the bubble sort, keys of adjacent items are compared and items are exchanged depending on the outcome of this comparison. After each complete pass through the items, at least one item is placed in its final position relative to all the other items. In the succeeding passes there is no need to examine this item again. Therefore, after each pass, the number of items to be examined is decreased by one. Moreover, it is not necessary to go through all the passes to completely sort the items. If, during any pass, no exchanges of items are performed, all items are already sorted. No further passes are necessary.

The sorting algorithms for the three models of data rearrangement operations are written in an ALGOL-like language and are presented in Appendix B. A sequence of basic data rearrangement operations is represented by the concatenation of the letters representing the operations, in the order in which they must be applied. For example, (abc) means the successive application of operations (a), (b) and (c). Exponentiation is used to represent repeated application of a sequence of operations, as in $(ab)^k$, meaning sequence (ab) applied k times.

4.1.1. Sorting in the First Model

The basic sort algorithm for the first model is essentially the "cocktail shaker" sort [34], a modified version of the bubble sort, in which alternate passes are taken in opposite directions. Passes in one direction seek the largest key and passes in the opposite direction seek the smallest key among the items to be examined. No auxiliary memory is needed for temporary data. However, three $\log_2 N$-bit counters are used for control purposes. One counter is used to hold the number of items to be examined in each pass, the second to control the length of a pass and the third counter is used to keep track of the number of passes. In addition to these counters, one single bit register is used to signal the end of the sorting process (detected by the abscence of exchanges during a pass). Algorithm B.1, described above, is presented in Appendix B.

The fact that the "cocktail shaker" sort is more efficient than the bubble sort enables us to use the results of bubble sort analysis to establish an upper bound on the average number of data rearrangement operations for Algorithm B.1. In the general case, the number of operations required to sort N items can be expressed as

$$t = (2NQ - Q^2 + 7Q - 2N - 4)/2 + 2X + R \quad \text{operations}$$

where Q is the number of passes, X is the number of item exchanges and R is the number of operations to move items to their final locations. R is equal to $(N - (Q-3)/2) \bmod N + 2$, if

Q is odd, and is equal to Q/2, if Q is even. From [34], we have that X(average) = $(N^2 - N)/4$ and Q(average) $< N + 1 - \sqrt{\pi N/2} + 2/3 - O(1/\sqrt{N})$. As R(average) $< N$, we get

$$t(\text{average}) < N^2 + 4.429N \text{ operations}$$

In the worst case, items initially in sorted order, Q is equal to N-1, X is equal to $(N^2 - N)/2$ and R is equal to (N-3)/2, and we get

$$t(\text{worst case}) = (3N^2 + 4N - 15)/2 \text{ operations}$$

The number of operations in Algorithm B.1 is dependent upon the number of exchanges and the number of passes. The only purpose of the last pass is the detection of the end of sort, as no exchanges are performed during the same. However, even if it were possible to detect the end of sort in the pass preceding the last one, the number of operations saved by not performing the last pass would be smaller than N, not significant when compared with the order of magnitude, N^2, of the algorithm. Another possibility of reducing the number of operations is to try to take advantage of some special configurations of the items during the sort process to decrease the length of passes. For example, see Figure 20, in which we show the sorting of 13 items. In passes 3 and 5 there is no need to compare items above the upper mark. The same is true with items below the lower mark, in pass 4. Moreover, pass 4 can start with the item below the upper mark obtained in pass 1, and pass 5 can start with the item above the lower mark obtained in pass 2. Hence, pass 3 can have its

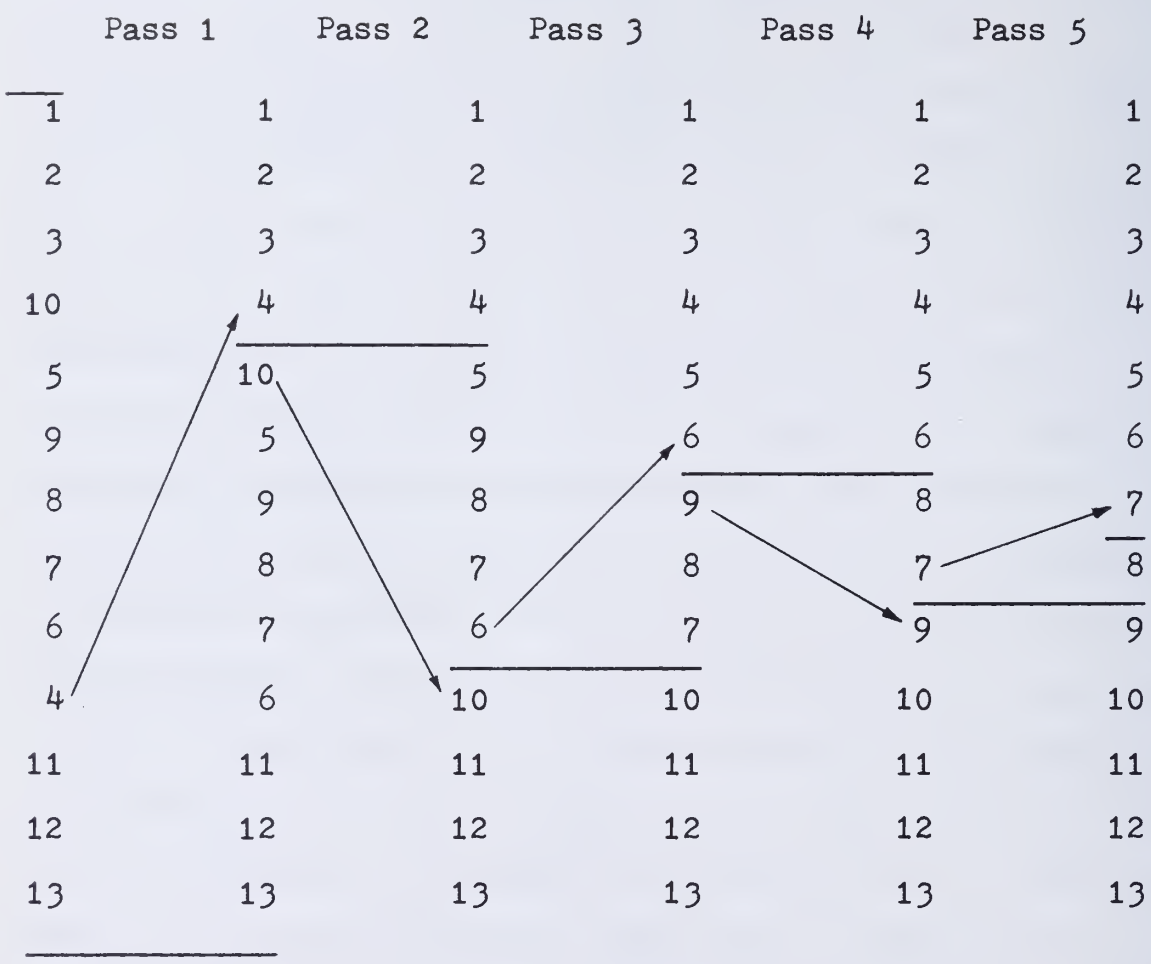


Figure 20. The modified "cocktail shaker" sort

length reduced by 3 operations, pass 4 by 6 operations and pass 5 by 7 operations. These configurations were detected by the abscence of exchanges for the corresponding items involved in passes 1, 3 and 2, respectively. In this particular example, the last pass is not needed and the total number of operations can be reduced to 85, as compared with the 103 operations needed in the original algorithm. Nevertheless, as the average number of exchanges is $O(N^2)$, again, the order of magnitude of Algorithm B.1 remains unchanged.

Optimality of the order of magnitude of Algorithm B.1 can be concluded from the results of Wong and Coppersmith [19]. More than that, we can claim that the algorithm is near optimum. This claim is based on the observations made by Knuth [35], when he addresses the one-tape sort problem by showing a connection between this problem and the "elevator problem". The "elevator problem", which can be stated as the problem of minimizing the number of moves to transport people between floors with a single elevator, by its turn, is connected to the "cocktail shaker" sort. The items are the people and the building is the loop containing them. The floors are the final locations in the loop and the elevator is location 0. In the analysis of the "elevator problem", the final destinations of people are assumed known (this is equivalent to assuming that the initial positions of the items in the loop are known).

4.1.2. Sorting in the Second Model

The basic algorithm for sorting in the second model is the simple bubble sort. Algorithm B.2, presented in Appendix B, also uses three $\log_2 N$-bit counters and one single bit register, and no auxiliary memory is needed for temporary data.

Analysis of Algorithm B.2 makes direct use of the results of bubble sort analysis. In the general case, the number of data rearrangement operations can be expressed as

$$t = NQ + 2X \quad \text{operations}$$

where Q is the number of passes and X is the number of exchanges. From [34], we have that $X(\text{average}) = (N^2 - N)/4$ and $Q(\text{average}) = N + 1 - \sqrt{\pi N/2} + 2/3 - O(1/\sqrt{N})$, and we get

$$t(\text{average}) = 3N^2/2 - N\sqrt{\pi N/2} + 7N/6 - O(\sqrt{N}) \quad \text{operations}$$

In the worst case, items in reverse order, Q is equal to N-1, and X is equal to $(N^2 - N)/2$, yielding

$$t(\text{worst case}) = 2N^2 - 2N \quad \text{operations}$$

The same observations made for Algorithm B.1, with respect to the last pass, are also valid for Algorithm B.2. However, there is no way to reduce the length of passes, due to the need in the latter algorithm, of shifting through items already sorted, after each pass. Nevertheless, the order of magnitude of the algorithm is also optimal, again from Wong and Coppersmith.

4.1.3. Sorting in the Third Model

The basic sort algorithm for the third model is essen-

tially Algorithm B.1, with slight modifications due to the different set of basic data rearrangement operations. Algorithm B.3 is presented in Appendix B.

The same observations made in the analysis of Algorithm B.1 are valid for Algorithm B.3. In the general case, the number of data rearrangement operations can be expressed as

$$t = (2NQ - Q^2 + Q)/2 + 2X + R \quad \text{operations}$$

where R is equal to (Q+3)/2, if Q is odd, and is equal to Q/2, if Q is even. Therefore, in the average case, as R(average) $<$ N/2, we get

$$t(\text{average}) < N^2 - 0.571N \quad \text{operations}$$

In the worst case, items initially in sorted order, we get

$$t(\text{worst case}) = (3N^2 - 3)/2 \quad \text{operations}$$

if N is odd, and

$$t(\text{worst case}) = 3N^2/2 \quad \text{operations}$$

if N is even.

As in Algorithm B.1, we can decrease the length of passes for special configurations of items in the loop, but, again, the order of magnitude of the number of operations in Algorithm B.3 remains unchanged.

4.1.4. Better Sort with Modified Models

Algorithms B.1, B.2 and B.3 have average time complexities of $O(N^2)$, $O(1.5N^2)$ and $O(N^2)$ operations, respectively. We already know that the order of magnitude of the number

of operations of these algorithms can not be reduced. However, it is possible to reduce the coefficients of N^2. One of the sources of inefficiency of the algorithms is that two extra operations, in addition to the shift of items, are needed to exchange items. These two operations are needed because item A (see Figure 7) is one location past the access port (the cell used as "pivot" in the operations), at the time comparison is done. In the first model, this suggests having the access port be location 1 instead of location 0, keeping location 0 as "pivot". Exchanges can now be performed by simply applying operation (b), saving two operations per exchange. With this modification in the first model, the number of operations required to completely sort N items, in the general case, is

$$t = (2NQ - Q^2 + 7Q - 4)/2 + R \quad \text{operations}$$

For the average and worst cases, the expressions for the number of operations are, respectively,

$$t(\text{average}) < N^2/2 + 4.929N \quad \text{operations}$$

and

$$t(\text{worst case}) = N^2/2 + 3N - 15 \quad \text{operations} \qquad (4.1)$$

This last result, Eq. (4.1), implies that

$$t(\text{average}) < N^2/2 + 3N - 15 \quad \text{operations}$$

The results above are very close to the estimate of the average number of operations for generation of arbitrary permutations ($0.4N^2$ operations) in the first model [19].

No improvement on the number of operations can be

attained, in the second model, by introducing the same modification. An improvement can be achieved, however, if we have operation (b) instead of operation (e) in the model. Again, having the access port be location 1 instead of location 0, we have, for the general case,

$t = NQ$ operations

and, for the average and worst cases, we have respectively,

$t(\text{average}) = N^2 - N\sqrt{\pi N/2} + 5N/3 - O(\sqrt{N})$ operations

and

$t(\text{worst case}) = N^2 - N$ operations

In the third model, a modification having the same effect of saving two extra operations per exchange can be achieved by having two access ports, one in location 1 and the other in location N-1. In the general case, time complexity becomes

$t = (2NQ - Q^2 + Q)/2 + R$ operations

and,

$t(\text{average}) < N^2/2 - 0.071N$ operations

and

$t(\text{worst case}) = (N^2 + 2N - 3)/2$ operations

if N is odd, and

$t(\text{worst case}) = (N^2 + 2N)/2$ operations

if N is even.

An interesting result of the preceding discussion is

that Eq. (4.1) represents a tighter upper bound for the worst case generation of permutations in the first model (an earlier upper bound, $3N^2/4 + 1/4$, was reported by Wong and Coppersmith).

Although the sorting algorithms discussed so far for the three models (modified or not modified) have comparable time complexities in terms of number of data rearrangement operations, it should be pointed out that operations in the first model take much more time (serial shift of one cell) than operations in the other two models (parallel shift). Sorting in the second model is best suited for out application since time complexity is small when items are more or less in sorted order initially.

4.1.5. Sorting in Minor Loops

We now consider sorting in magnetic bubble memories when each cell in a loop of N cells is the set of identically numbered positions in the minor loops of a chip with major-minor loop chip organization. (The possibility of implementing the second and third models in the minor loops of the major-minor loop chip organization was mentioned in Section 2.2.). In this case, as with the bubble ladder implementation of the first or third models, the loop of cells is implemented in a single chip. Moreover, although data rearrangement operations still require only one bubble shift, comparison of keys requires that the contents of entire cells are read out bit serially. Consequently, as the major and minor loops in a chip are shifted simultaneously, several extra shifts could be necessary to make the items

being read out be at the correct positions when data rearrangement operations are applied. This synchronization demands no extra shifts (or very few extra shifts) if the number of minor loops, L (= W), is a multiple (or very close to a multiple) of the minor loop length, S (= N).

Assuming that the synchronization above requires no extra shifts, it is simple the computation of the time complexities of sorting in terms of bubble shifts in the second and third models. It suffices, for the respective sorting algorithms, to add the number of comparisons multiplied by W to the number of operations required to completely sort the items. From [34], we get, for algorithms B.2 and B.3, respectively,

$$t(\text{average}) = (N^2W - NW\log N - 0.171NW + 3N^2)/2 + O(W\sqrt{N})$$

shifts and

$$t(\text{worst case}) = W(N^2 - N)/2 + 2N^2 - 2N \text{ shifts}$$

and,

$$t(\text{average}) < (N^2W - NW\log N - 0.171NW + N^2)/2 \text{ shifts}$$

and

$$t(\text{worst case}) \quad (W(N^2 - N) + 3N^2)/2 \text{ shifts}$$

where log stands for the natural logarithm and 0.171 is an approximation for an expression involving the constant γ.

4.2. Merging in Bubble Memories

A very simple algorithm, illustrated in the example in Figure 21, may be used to merge the contents of two data loops.

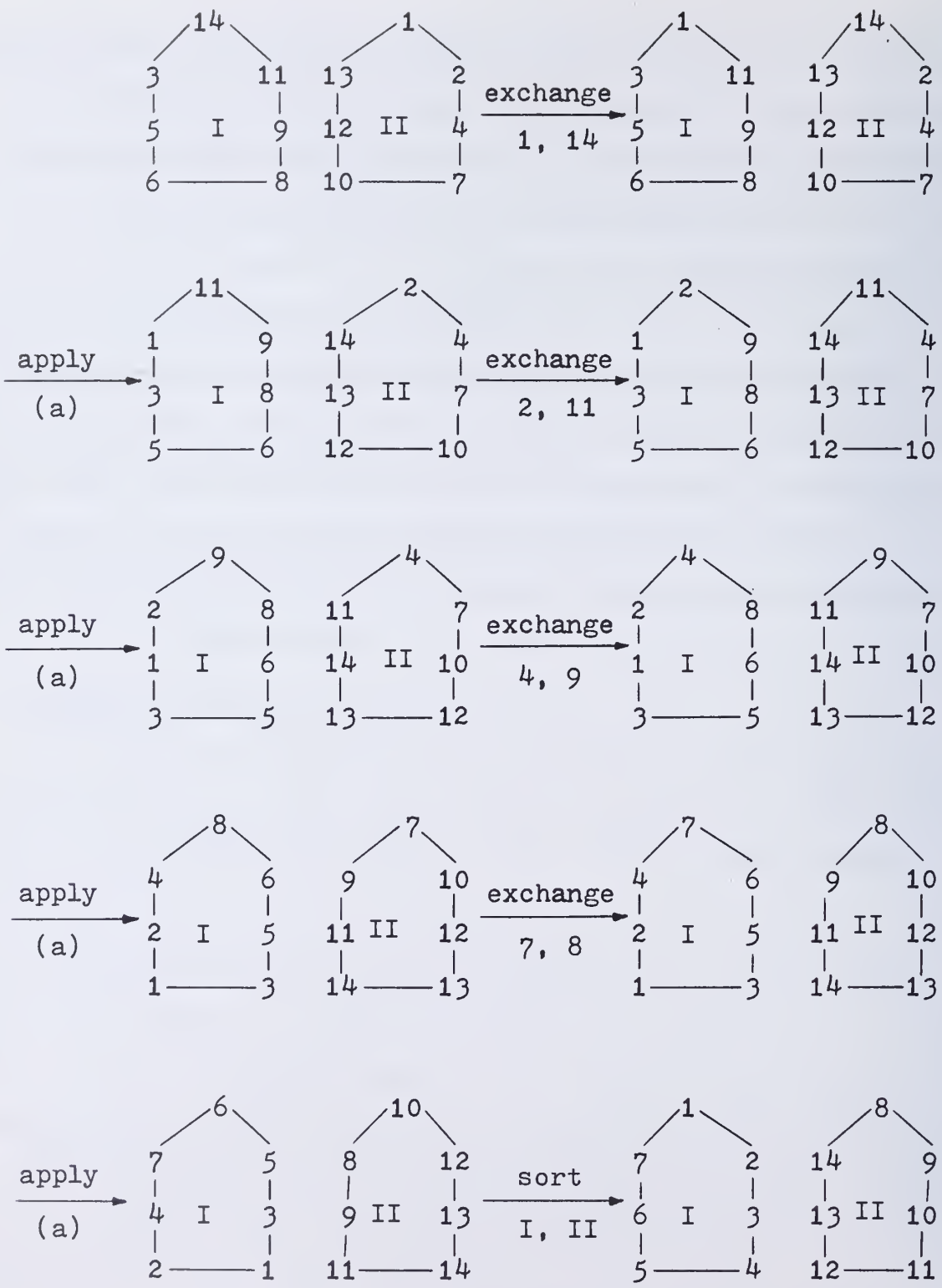


Figure 21. An example of merging two data loops.

Specifically, we want to merge the contents of two N-cell loops so that, after merging, the smallest N items are in one loop (say, loop I) and the largest N items are in the other loop(loop II). Contents of the loops may then be sorted independently. In this algorithm, described informally below, we assume that the contents of loops I and II are initially sorted in non-increasing and non-decreasing order, respectively. (If data rearrangement operation (f) is available, both loops may be sorted initially in non-decreasing order.) Keys of the two items at the access ports (say, A in loop I and B in loop II) are compared (bit serially in the first model and possibly in parallel in the second and third models). When the value of A is found to be larger than the value of B, the two items are exchanged and operation (a) is applied to both loops (or operation (f) in loop I and operation (a) in loop II). When the value of A is found to be smaller than or equal to the value of B, loop I contains the N smallest items.

For the evaluation of the time complexity of this algorithm, we assume that all configurations of items in the two loops are equally probable. From a simple combinatorial argument, we have that the number of operations, on the average, can be expressed as

$$t(\text{average}) = \sum_{i=1}^{N} (N!)^2 \binom{N}{i}^2 i \Big/ \sum_{i=1}^{N} (N!)^2 \binom{N}{i}^2$$

which simplifies to

$$t(\text{average}) = N/2 \quad \text{operations}$$

In the worst case, we have initially the N smallest items in loop II and the N largest items in loop I, and we get

t(worst case) = N operations

These same expressions above are valid for the number of comparisons and the number of item exchanges required to merge two loops of N cells each.

Merging K N-cell loops without auxiliary storage for temporary results requires alternating passes of merging contents of pairs of loops and sorting the contents of individual loops as illustrated in Figure 22. After merging, loop I contains the smallest N items, loop II contains the next N smallest items, etc. When K is a power of two there is one merging pass involving one pair of KN/2-cell "loops", there are two merging passes involving two pairs of KN/4-cell "loops", etc, K/4 merging passes involving K/4 pairs of 2N-cell "loops" and K/2 merging passes involving K/2 pairs of N-cell loops. These merging passes are alternated with passes of sorting the contents of individual loops. Therefore, there are K-1 merging and sorting passes. (If K is not a power of two, K_1-1 passes are necessary, where K_1 is the immediate power of two greater than K.) It is easy to show, by induction on K, that the time complexity of the merging algorithm above is $O(C_1(K-1)N^2 + C_2(KN\log_2 K)/2)$ operations plus $O(C_2(KN\log_2 K)/2)$ exchanges, where the values of the constants C_1 and C_2 depend on the particular model of data rearrangement operations. In the modified first model, the worst case values of C_1 and C_2 are 1/2 and 1, respectively. Since

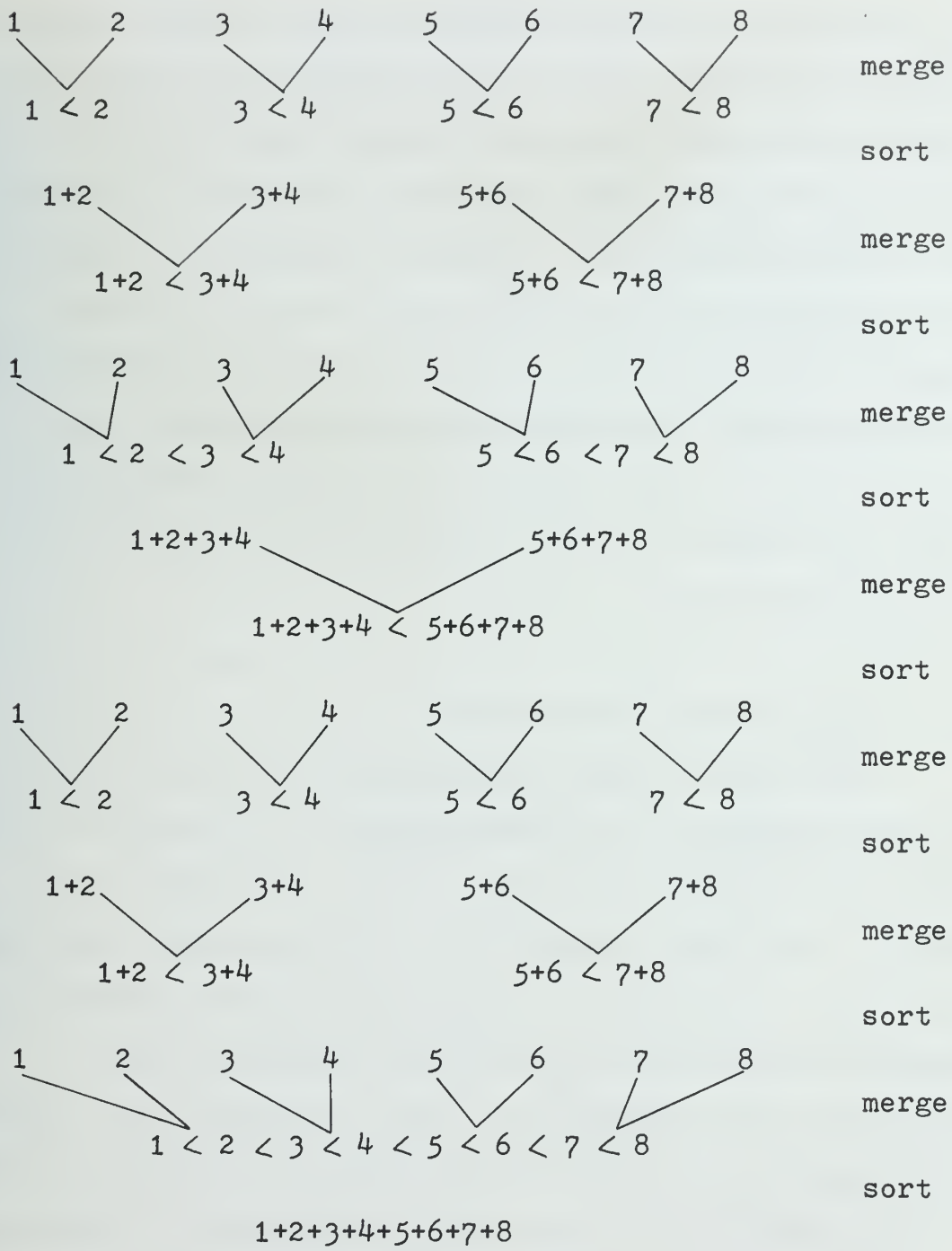


Figure 22. Merging 8 loops without extra storage.

each operation and each item exchange take the time for W shifts, the worst case merging time is $O((K-1)N^2W/2 + KNW\log_2 K)$ shifts. On the other hand, when the second model is implemented in the minor loops of major-minor loop chips, the time for the sorting passes is spent mostly with reading out items for comparison of keys and the worst case merging time is $O((K-1)(N^2-N)W/2 + KNW\log_2 K)$ shifts. However, if the second model is implemented in the major loops, the worst case merging time is $O(2(K-1)(N^2-N) + KN\log_2 K)$ shifts, as items are read out and written bit parallelly for the comparison of keys and to be exchanged, respectively.

4.2.1. Merging with Extra Storage

The time required for merging can be reduced by providing extra storage for temporary results and by using multiway comparators. In particular, when a K-way comparator (which selects as its output the smallest of its K inputs, like the merge network described by Hollaar [36]) is used together with one extra storage loop, the algorithm described below can be used to merge the contents of K loops. We assume that initially all K loops are sorted in non-decreasing order. Passes of transfer of loop contents are alternated with passes of sorting contents of individual loops. In the first pass, the N smallest items, obtained as the outputs of the K-way comparator, are transferred to the extra loop. The contents of loop I are then transferred to empty cells in loops II, ..., K. These loops are

then sorted, and the same procedure is applied with loop I as the extra loop. In the next iteration, loop II is the extra loop and so on, until there are only two loops (K-1 and K) left to be merged. Finally these two loops are merged and the contents of loop K are transferred to loop K-1. After these K-1 iterations of the algorithm, the loop that started as the extra one contains the N smallest items, loop I contains the next N smallest, loop II the next N smallest, etc, and loop K is empty. The time complexity of this algorithm is $O(C_1(K-1)N^2 + (1+C_3)(K-1)N)$ operations plus $O((1+C_3)(K-1)N)$ item transfers, where the value of the constant C_1 depends again on the particular model of data rearrangement operations and the value of C_3 depends on the way items are distributed among the loops. In the modified first model, the worst case merging time is $O((K-1)N^2W/2 + 4(K-1)NW)$ shifts. For implementations of the second model in the minor loops and in the major loops the worst case merging times are respectively, $O((K-1)(N^2-N)W/2 + 4(K-1)NW)$ shifts and $O((K-1)(N^2-N)/2 + 4(K-1)N)$ shifts.

When E extra storage loops are available for temporary results, the merging algorithm is essentially the same and its time complexity is $O(C_1(K/E - 1)N^2 + (K-E)N + C_4(K/E - 1)N)$ operations plus $O((K-E)N + C_4(K/E - 1)N)$ item transfers. (In each iteration of the algorithm, the transfer and shifting of the EN smallest items is done one item at a time and the emptying of loops can be done in parallel.) Results for the different models of data rearrangement operations are similar to the ones

in the case of one single extra loop.

The sorting and merging algorithms discussed so far allow us to sort a file into N-item buckets. Most of the time required in this process is spent in sorting the contents of individual loops. (Items are moved among loops only during $O(NK\log_2 K)$ operations when no extra data loops are provided and $O(NK)$ operations when extra data loops are provided during merging.) Retrieval from the file can clearly be allowed during sorting. In Table 6, we summarize the complexities of the various algorithms in different memory organizations.

4.3. Clustering in Bubble Memories

Clustering is the process of rearranging items according to a set of mutually exclusive properties so that items sharing the same property are grouped together in the data loop. An ordering can be defined on the set of properties and the clustering process can then be reduced to a sorting or merging process where the values of the keys are determined by this ordering. Therefore, the same algorithms designed for sorting and merging can be used for the clustering of items stored in a data loop. Furthermore, it seems doubtful that algorithms for clustering making use of the three models of data rearrangement operations exist which are much more efficient than the ones presented in this chapter for sorting and merging.

Table 6. Merging Complexities in Different Memory Organizations. Worst case expressions in number of bubble shifts for K N-cell buckets, W-bit cells.

Algorithm/ Model	Implementation	Without Extra Loops	E Extra Loops	Sorted Buckets
Gyro Sort	Edelberg and Schissler [11]	$KN^2W/2 + NW$		No
Modified First Model	Bubble Ladder	$KN^2W/2 + KNW(\log_2 K+3) - 15KW$ (*)	$\lceil K/E \rceil N^2W/2 + KNW + 4\lceil K/E \rceil NW - 15\lceil K/E \rceil W$	Yes
Second Model	Minor Loops in Major-minor Loop	$K(N^2-N)W/2 + KNW\log_2 K + 2K(N^2-N)$ (*)	$\lceil K/E \rceil N^2W/2 + NW(K+\lceil K/E \rceil /2) + 2\lceil K/E \rceil (N^2-N)$	Yes
	Major Loops in Major-minor Loop	$2K(N^2-N) + KN\log_2 K$ (*)	$2\lceil K/E \rceil N^2 + KN + \lceil K/E \rceil N$	Yes

(*) These expressions are valid when K is power of two.

4.4. Comparison to Previous Results

There are several recent papers dealing with sorting in magnetic bubble memories. Wong and Coppersmith [19] designed optimal algorithms for generating arbitrary permutations that can be used for sorting data in a magnetic bubble memory. It is shown that the time complexity of the algorithms is $O(N^2)$, where N is the number of cells. However, no comparison of keys is mentioned in the development of the algorithms. Therefore, it is assumed that the initial positions of items in the data loop are known. Moreover, the algorithms do not include the operations required in moving items to the access port for the comparison of keys. A sorter device based on the odd-even transposition sort with multiple shift register loops of fixed size (cells, in our terminology) has been proposed [21]. In this case, a special processor with N-1 comparators (N is the number of cells) performs the parallel comparison of keys and controls the sorting process through switches linking the cells. Sort can be completed in O(N) periods, one period being the time for the serial shift of one entire cell. Keys must be available to the processor, either by storing them separately from the items or through access ports provided in each of the cells.

The memory systems and elementary file processing algorithms have performance comparable to the intelligent memory proposed by Edelberg and Schissler [11]. The memory described in [11] contains K N-cell loops (W bits per cell) with two access ports for serial input/output. K processing elements are

required to sort the contents of the memory into K buckets. Each bucket contains N unsorted items. In the worst case, the time complexity of their sorting algorithm, called "gyro" sort, is $O(KN^2W/2)$ shifts. The "gyro" sort complexity is also listed in Table 6.

5. PHYSICAL STORAGE ORGANIZATION OF RELATIONAL DATA BASE

We are concerned here with the various schemes to store relations and tuples from relations in magnetic bubble memories and with the design of algorithms for primitive relational algebraic operations needed to support directly the relational data model.

An example of a data base machine architecture in which intelligent magnetic bubble memories are used to store relations and process them under the supervisory control of the primitive operations control processor (POCP) is described in Chapter 1. The intelligent memory itself is composed of control units, a memory interface, and magnetic bubble memory modules. As shown in Figure 23, each control unit controls directly a fixed number of memory modules and the corresponding section of the memory interface. The memory interface contains comparators and a selector whose function is explained below. Tuples are transferred to and from memory modules through the memory interface.

Comparison of tuples or domain values is performed in two steps. A pre-comparison in the memory interface yields partial results of the comparison of W-bit segments of tuples (W is a design parameter and its value represents the minimum amount of storage assigned to a domain). According to the domains required by the operation being performed, the outputs of the corresponding comparators are selected to be the input to the comparator in the control unit. The output of the latter comparator is used by the control unit to decide which further

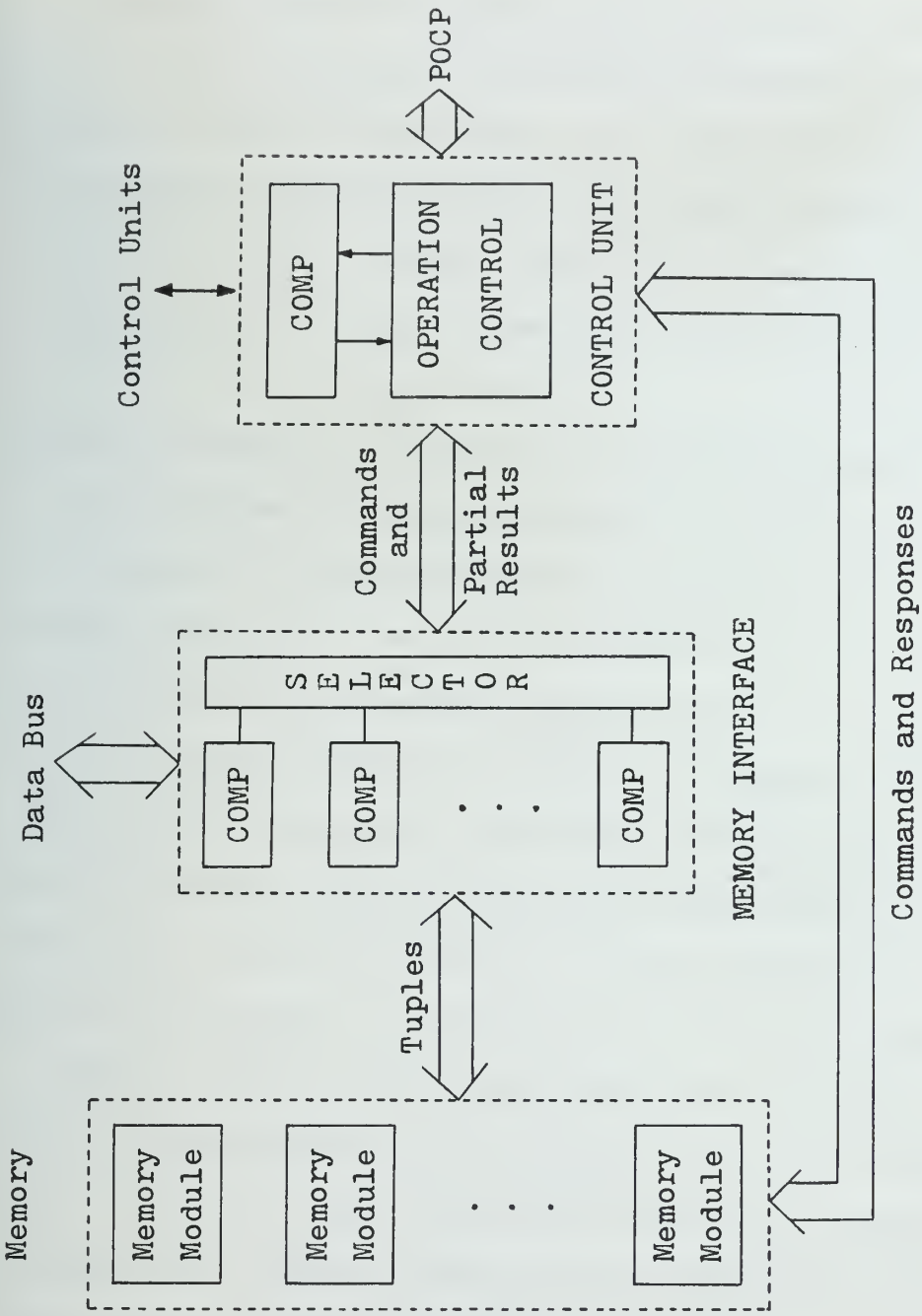


Figure 23. Architecture of an intelligent magnetic bubble memory for relational operations.

command(s) is to be issued to memory modules. (An alternative arrangement for the memory interface is to have the selection of the domains performed before the comparison of their values. In this case, a smaller number of possibly shorter comparators in the memory interface are required. However, the time for comparisons is variable and, thus, more complex control units are needed.)

The overlapping control organization of memory modules, shown in Figure 24, allows a certain degree of flexibility in the processing of relations. Each memory module is assigned to two control units and each control unit has a total of M memory modules assigned to it (M is a design parameter). Therefore, up to three relations stored in a group of M memory modules can be processed simultaneously, by one control unit and its two immediate neighbors. Furthermore, with a mechanism provided for the synchronization of control units, the maximum tuple length that can be processed is limited only by the way relations are stored in the memory modules and by the amount of available storage.

Implementations of models of data rearrangement operations were proposed using the major-minor loop bubble chip organization and the uniform bubble ladder. These chip organizations offer the possibility of design options for relational storage organization with variable degree of concurrency of access to relations, time efficiency of relational operations, control complexity, and amount of unused space due to fragmentation in bubble chips. We measure the degree of concurrency

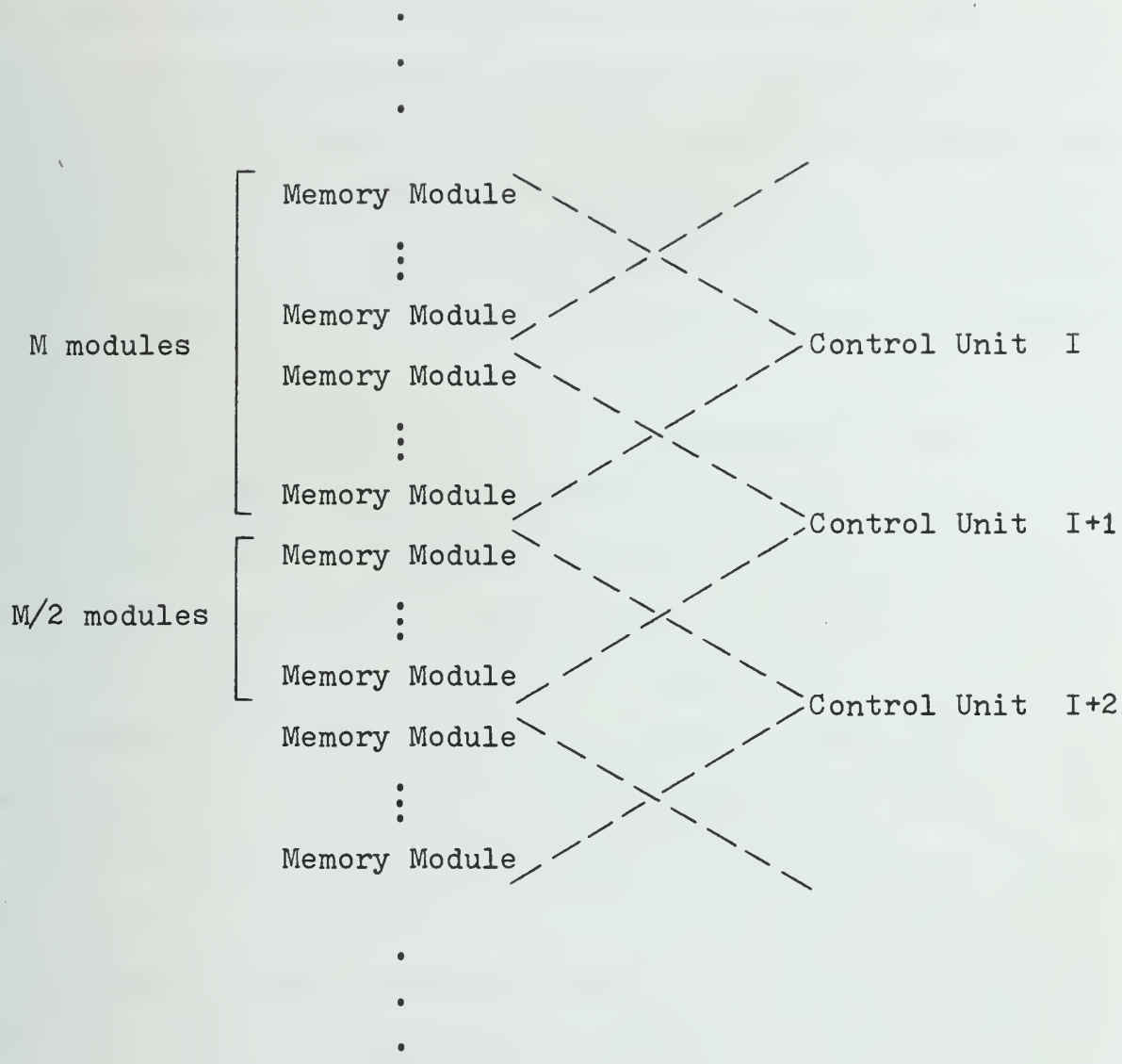


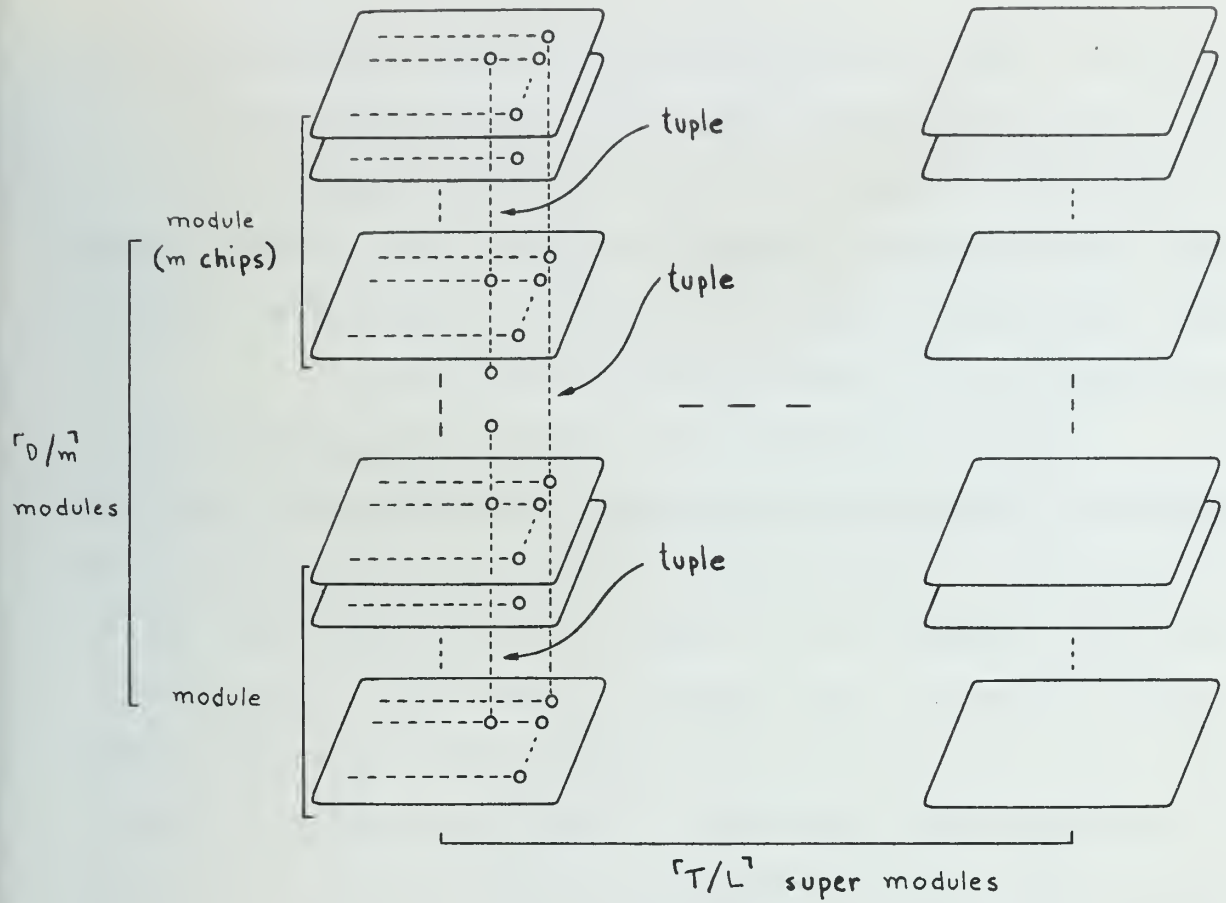
Figure 24. Overlapping control organization of modules.

by the number of relations or the fraction of the data base that can be independently accessed. The time complexity of the sort-merge algorithm of Chapter 4 is used to evaluate the efficiency of operations in each organization. Control complexity is measured in terms of the extra storage and of the number, size and type of comparators needed to perform the operations.

5.1. Storage Organizations

Again, we consider a magnetic bubble memory containing C B-bit chips, with each chip containing L S-bit minor loops or, in the case of the bubble ladder, S L-bit cells. The chips are grouped into independent modules each containing m chips. Among all possible schemes for storing a relation containing T D-bit tuples in the memory, we consider only the following three storage organizations. With these three storage organizations, sorting and merging operations can be performed autonomously in the memory modules without very complicated control mechanisms.

(i) Storage organization I. Chip shared. In this case, each chip contains only one bit from any tuple. A super module composed of $\lceil D/m \rceil$ modules is used to store a tuple. Many relations may share these $\lceil D/m \rceil$ modules. As shown in Figure 25, bits from up to S relations are stored in a chip and tuples of a relation are distributed among minor loops so that L tuples of the relation can be brought in parallel to the major loops. These $mL\lceil D/m \rceil$ bit positions in a super module constitute what we call a bucket. Each relation is assigned an integer number of buckets



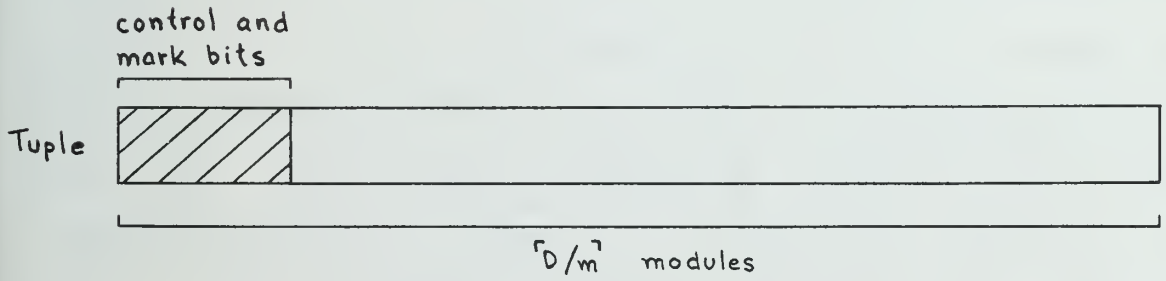


Figure 25. Storage Organization I. Chip shared.

and modules. That is, relations do not share buckets in the same module. Tuples in a super module are accessed bit parallelly. When $\lceil D/m \rceil \lceil T/L \rceil$ modules are used to store a relation, $\lceil T/L \rceil$ tuples can be accessed in parallel. Feasibility of data rearrangement operations in major loop only is assumed. The uniform bubble ladder chip organization can not be used.

(ii) Storage organization II. Single relation per chip/segment per chip/segment distributed. In this case, relations are assigned an integer number of memory modules. A module contains L-bit segments from S tuples of a relation. Each chip in a module stores one L-bit segment of each tuple, distributed among the L minor loops or stored in a cell of a bubble ladder. In the former case, feasibility of data rearrangement operations in minor loops is assumed. As shown in Figure 26, a tuple is stored in a super module composed of $\lceil D/Lm \rceil$ modules. $\lceil T/S \rceil \lceil D/Lm \rceil$ modules are used to store a T-tuple relation. Each L-bit segment of a tuple is accessed bit serially and $\lceil D/L \rceil$ segments of a tuple can be accessed in parallel. Moreover, $\lceil T/S \rceil$ tuples in a relation can be accessed in parallel.

(iii) Storage organization III. Single relation per module/bit per chip. In this case, bits from a tuple are stored in a module in the same way as in organization I. However, each module contains tuples from only one relation. Again, feasibility of data rearrangement operations in major loop only is assumed and the uniform bubble ladder chip organization can not be used. As shown in Figure 27, a super module composed of

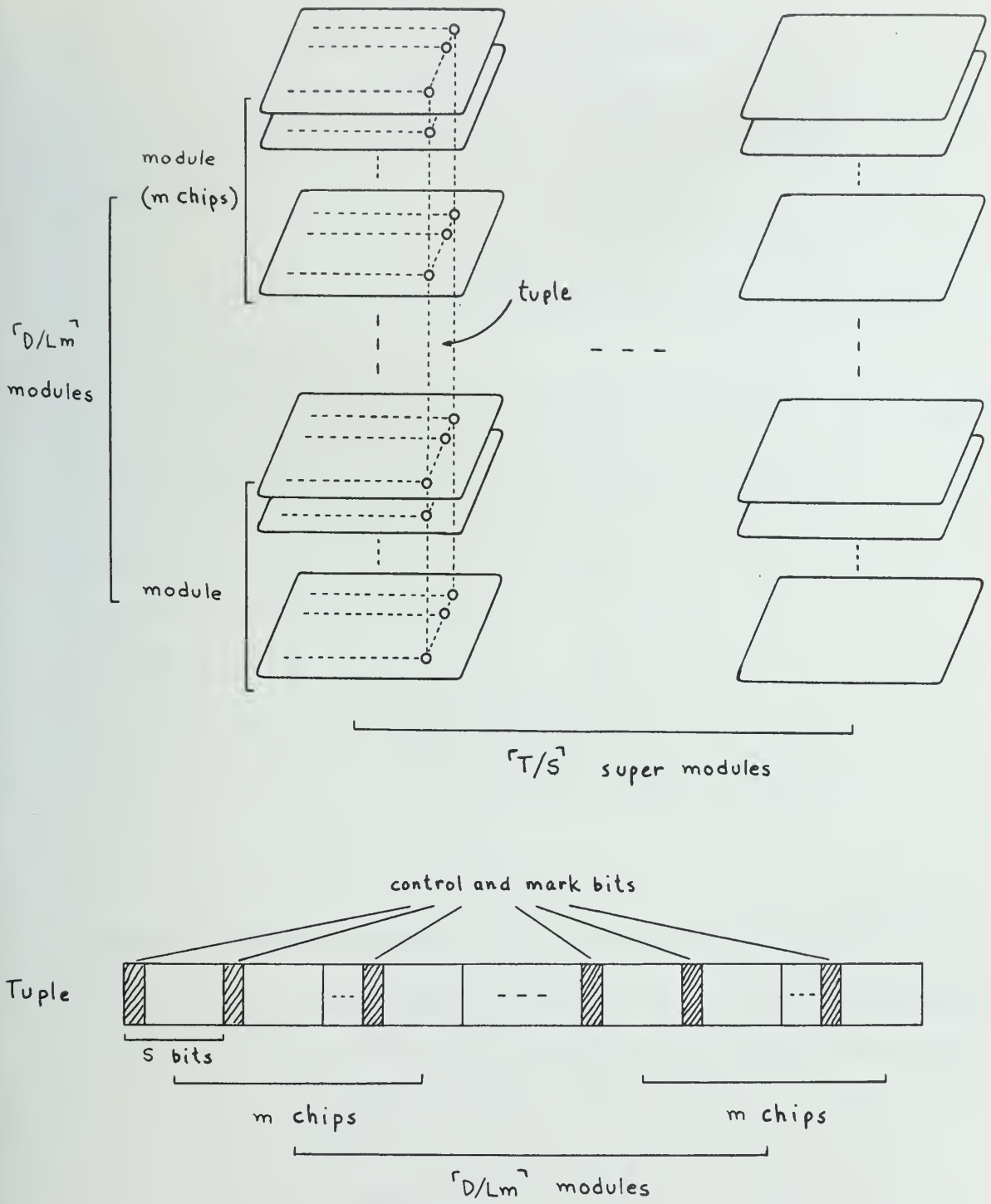


Figure 26. Storage Organization II. Single relation per chip/segment per chip/segment distributed.

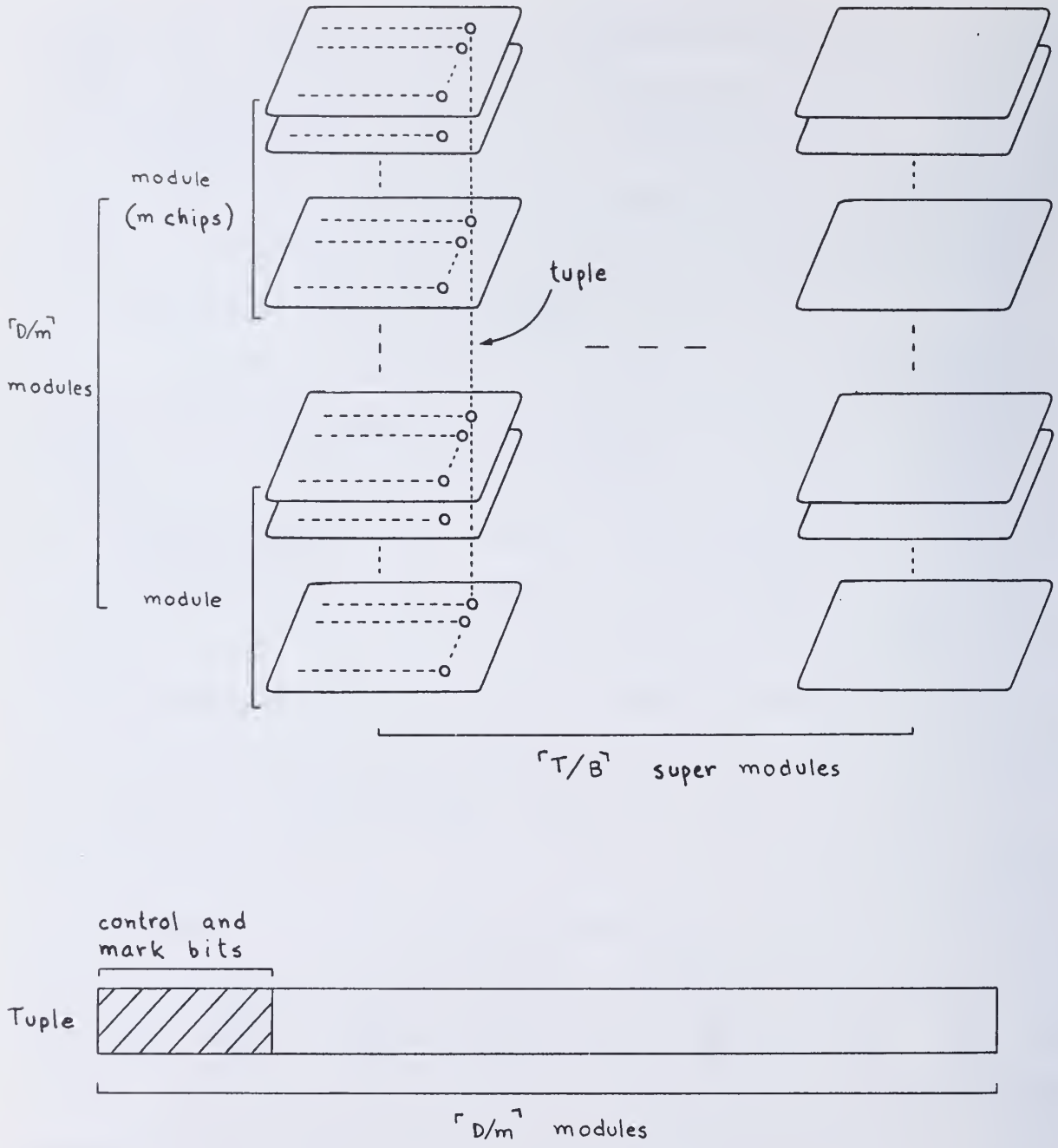


Figure 27. Storage Organization III. Single relation per module/bit per chip.

$\lceil D/m \rceil$ modules is used to store a tuple and $\lceil D/m \rceil \lceil T/B \rceil$ modules are required to store a relation. Again, each tuple is accessed bit parallelly but only $\lceil T/B \rceil$ tuples of a relation can be accessed in parallel.

5.2. Control Complexity, Degree of Concurrency and Efficiency of Operations

In organization I, in order to take full advantage of the bit parallel access of tuples, C/W W-bit parallel comparators are needed in the memory interface. Each of these comparators yields a two-bit output. Hence, one (2Mm/W)-bit parallel comparator is needed in each of the 2C/Mm control units in the data base. (For example, for C = 4096, M = 128, m = 8 and W = 16, 256 16-bit and 8 128-bit parallel comparators are necessary. Alternative schemes using a smaller number of shorter comparators are possible. In such schemes, values of selected domains are entered directly into the control units. A single bit-parallel comparator (whose size is a design parameter) in each control unit performs the comparisons required by the operation being performed. However, control units are more complex and comparison times (outside the memory modules) are larger in these schemes. For example, for the same values of C, M, m, and W above, only 8 16-bit parallel comparators are necessary but comparison time outside memory modules can be up to 64 times slower than in the example above.) CB/S bits of extra storage are needed if operations between relations stored in the same super module are to

be allowed, and CB/SL bits of extra storage, otherwise.

Variable relation and tuple sizes, in addition to the sharing of super modules by relations, make it difficult to determine an expression for the number of relations that can be accessed concurrently in organization I. However, it is clear that 1/S of the data base can be accessed concurrently. From Chapter 4, we have that the time complexity to sort-merge a relation is O(TL) bubble shifts. We note that there is a trade-off between degree of concurrency and efficiency of operations, as illustrated in Figure 28, in which we plotted S as a function of L for fixed size ships (SL = constant). Efficiency of operations and degree of concurrency are inversely proportional to L and S, respectively.

In organization II, in order to take advantage of the L-bit segment parallel access to tuples, C bit serial comparators are needed in the memory interface. A comparison in the memory interface takes L bubble shifts. Each W-bit segment comparison yields a two-bit output for the selector. Clearly, it is more realistic to have one bit serial comparator instead of one (2MLm/W)-bit parallel comparator in each of the 2C/Mm control units in the data base. (For example, for C = 4096, M = 2, m = 8, W = 16 and L = S = 64, in addition to the 4096 bit serial comparators in the memory interface, 512 bit serial comparators are necessary in the control units, in the former case, instead of the 512 128-bit parallel comparators needed in the latter case.) CB/S bits of extra storage are needed.

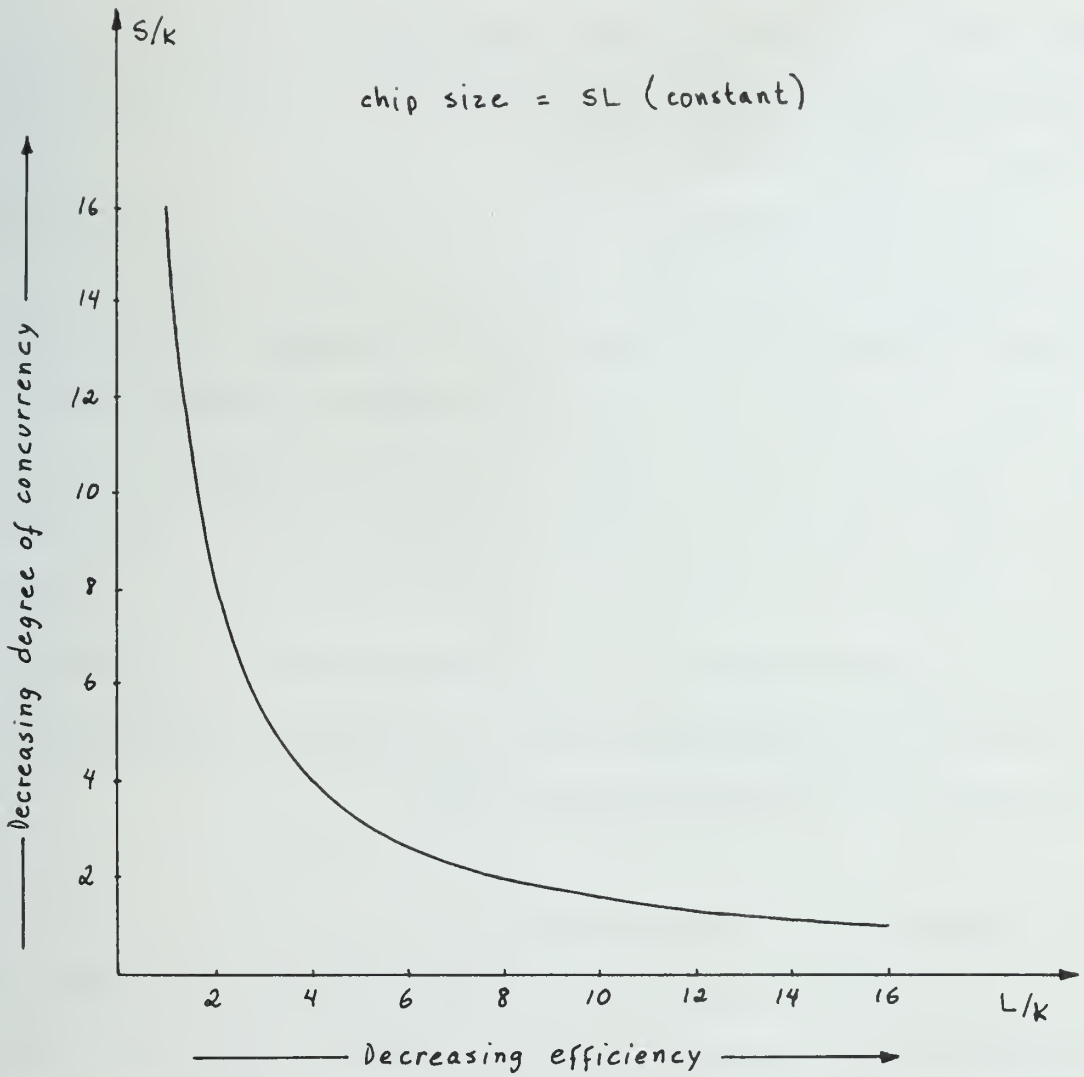


Figure 28. Degree of concurrency versus efficiency in storage organization I.

All relations in the data base can be accessed concurrently. If there are N relations in the data base, up to N concurrent users can be supported in this storage organization. Again from Chapter 4, we have that the time complexity to sort-merge a relation is O(TSL) bubble shifts.

For organization III, the same results of organization I, with respect to the number, size and type of comparators, are valid. However, CB/S bits of extra storage are needed. Furthermore, although each group of L tuples which can be brought in parallel to the major loops can be sorted in $O(L^2)$ bubble shifts, merging of these groups of L tuples stored in the same super module requires $O(S^2L^2)$ bubble shifts. Therefore, sort-merge complexity of a relation stored in $\lceil T/SL \rceil$ super modules is O(TSL) bubble shifts. Again, all relations in a data base can be accessed concurrently.

5.3. Fragmentation in Bubble Chips

In storage organization I, fragmentation can occur for the following two reasons: (1) Tuple length, D, is not a multiple of the module size, m, in which case one module in each super module is not completely filled. (2) Number of tuples, T, is not a multiple of the number of minor loops, L, in which case one bucket is not completely filled. The cases where entire buckets or modules in a super module are empty are not considered as fragmentation, as these buckets or modules can eventually be used to store other relations. The amount of fragmen-

tation correspondent to one relation can be derived as being

$$f = \lceil T/L \rceil L \lceil D/m \rceil m - DT \quad \text{bits}$$

Assuming that the average tuple length is D_A and the average number of tuples is T_A, the average amount of fragmentation per relation can be written as

$$f(ave) = (T_A + E(\lceil T/L \rceil L - T))(D_A + E(\lceil D/m \rceil m - D)) - D_A T_A \quad \text{bits}$$

where $E(X)$ is the expected value of X. The unused space to useful storage ratio is given by

$$r = f(ave)/D_A T_A = E(\lceil D/m \rceil m - D)/D_A + E(\lceil T/L \rceil L - T)/T_A + E(\lceil D/m \rceil m - D)\, E(\lceil T/L \rceil L - T)/ D_A T_A \qquad (5.1)$$

In the case where both the number of tuples and the tuple length are uniformly distributed random variables, we have

$$r(unif) = m/2D_A + L/2T_A + mL/4D_A T_A$$

In Figure 29 we plotted the above ratio as a function of D_A and T_A, expressed as submultiple and multiple of m and L, respectively. As expected, better storage utilization is achieved for large values of D_A and T_A as compared to the number of chips in a module, m, and the number of minor loops in a chip, L, respectively.

If the number of tuples and the tuple length are exponentially distributed random variables, we have first to determine the expression for $E(\lceil I/a \rceil a - I)$, where I is an exponentially distributed random integer variable with average

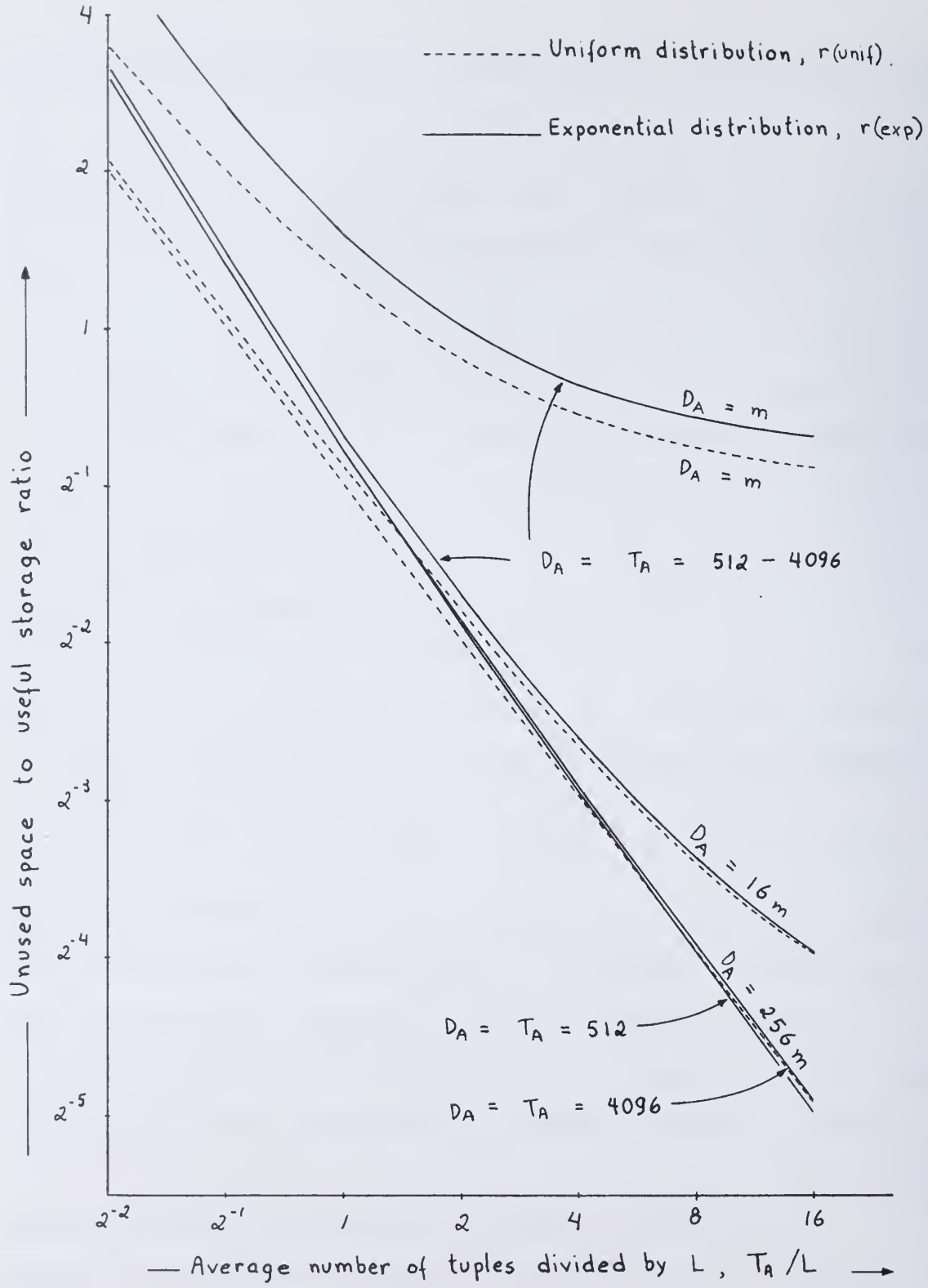


Figure 29. Unused space to useful storage ratio for chip shared storage organization.

I_A and a is an integer constant. It can be written as

$$E(\lceil I/a \rceil a - I) = a \exp(a/I_A)/(\exp(a/I_A) - 1) - \exp(1/I_A)/(\exp(1/I_A) - 1) \qquad (5.2)$$

Substituting (5.2) into (5.1), we get

$$r(exp) = d/D_A + t/T_A + dt/D_A T_A$$

where

$$d = m \exp(m/D_A)/(\exp(m/D_A)-1) - \exp(1/D_A)/(\exp(1/D_A)-1)$$

and

$$t = L \exp(L/T_A)/(\exp(L/T_A)-1) - \exp(1/T_A)/(\exp(1/T_A)-1)$$

From the curves plotted in Figure 29, we can see that r(exp) also decreases for large values of D_A and T_A, as compared to m and L, respectively. Furthermore, r(exp) is larger than r(unif) for comparable relative values of D_A and m, and, T_A and L. However, for small values of D_A and m (see curves for $D_A = T_A = 512$ and $D_A = 256m$), r(exp) is greater (smaller) than r(unif) for small (large) values of T_A, relative to L. No appreciable effect of the absolute values of $D_A = T_A$ on r(exp) was observed in the range of values for which curves were plotted.

In storage organization II, fragmentation can occur for the following reasons: (1) Tuple length, D, is not a multiple of mL, in which case one module in each super module is not completely filled. (2) Number of tuples, T, is not a multiple of S, in which case one super module is not completely filled.

The amount of fragmentation correspondent to one relation can be determined to be

$$f = \lceil T/S \rceil \lceil D/mL \rceil SLm - DT \text{ bits}$$

The average amount of fragmentation per relation can be written as

$$f(ave) = (T_A + E(\lceil T/S \rceil S - T))(D_A + E(\lceil D/mL \rceil mL - D)) - D_A T_A \text{ bits}$$

and the unused space to useful storage ratio is given by

$$r = f(ave)/D_A T_A = E(\lceil D/mL \rceil mL - D)/D_A + E(\lceil T/S \rceil S - T)/T_A + E(\lceil D/mL \rceil mL - D)\, E(\lceil T/S \rceil S - T)/D_A T_A$$

where D_A and T_A are the average tuple length and average number of tuples, respectively. The unused space to useful storage ratios for D and T having uniform and exponential distributions are, respectively,

$$r(unif) = mL/2D_A + S/2T_A + mLS/4D_A T_A$$

and

$$r(exp) = d/D_A + t/T_A + dt/D_A T_A$$

where

$$d = mL\ \exp(mL/D_A)/(\exp(mL/D_A)-1) - \exp(1/D_A)/(\exp(1/D_A)-1)$$

and

$$t = S\ \exp(S/T_A)/(\exp(S/T_A)-1) - \exp(1/T_A)/(\exp(1/T_A)-1)$$

Curves plotted in Figure 30 for the ratios above are identical to the ones plotted for organization I. However, in Figure 30, r(unif) and r(exp) are plotted as functions of D_A/mL and T_A/S, which means that larger values of D_A are needed in organization II to achieve storage utilization comparable to that in organization I.

In storage organization III, fragmentation can occur for the following reasons: (1) Tuple length, D, is not a multiple of m, in which case one module in each super module is not completely filled. (2) Number of tuples, T, is not a multiple of B (= SL), in which case one super module is not completely filled. The amount of fragmentation correspondent to one relation can be determined to be

$$f = \lceil T/SL \rceil \lceil D/m \rceil mLS - DT \text{ bits}$$

and the average amount of fragmentation per relation can be written as

$$f(ave) = (T_A + E(\lceil T/SL \rceil SL - T))(D_A + E(\lceil D/m \rceil m - D)) - D_A T_A \text{ bits}$$

Similarly to organizations I and II, we have

$$r(unif) = m/2D_A + SL/2T_A + mSL/4D_A T_A$$

and

$$r(exp) = d/D_A + t/T_A = dt/D_A T_A$$

where

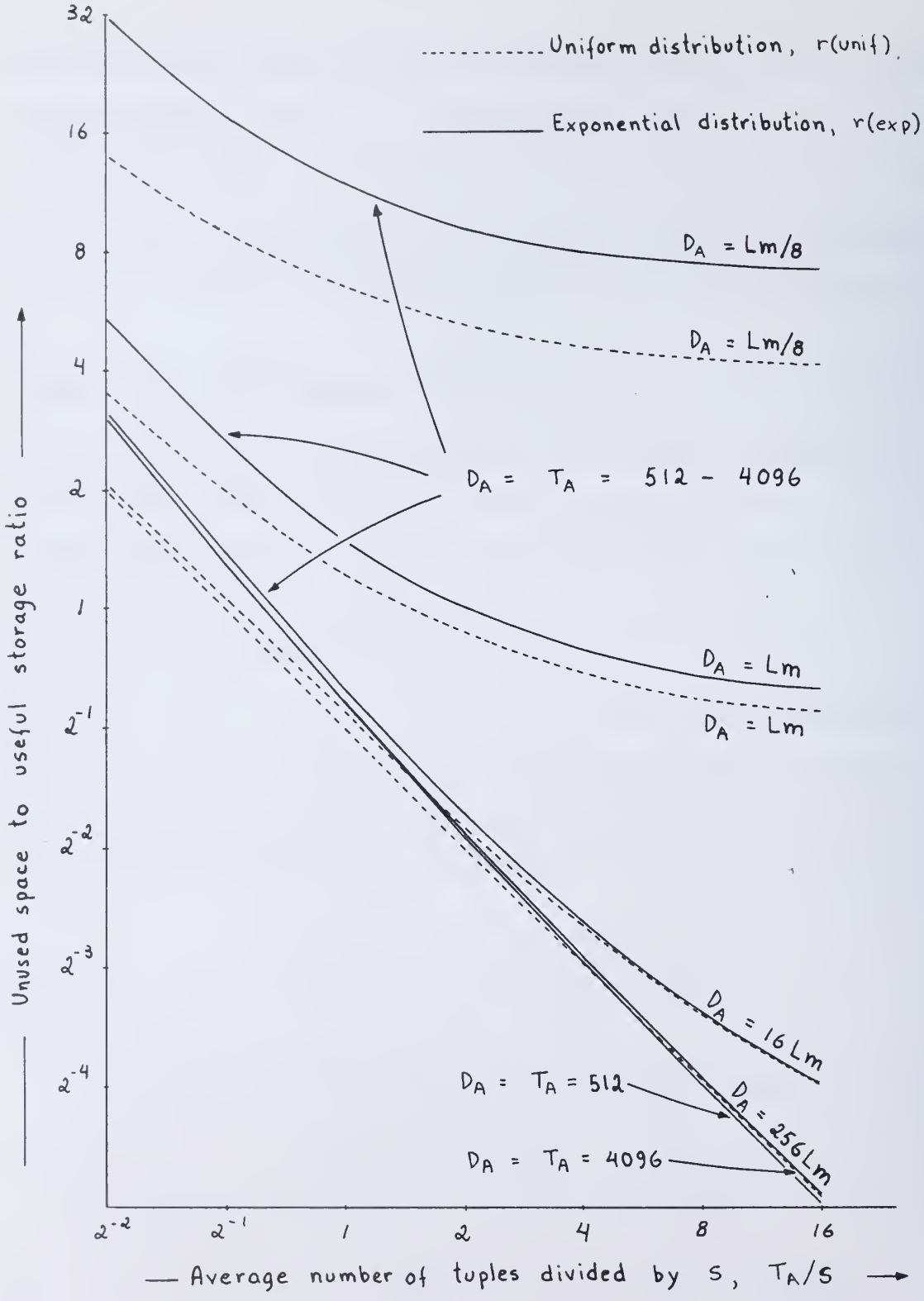


Figure 30. Unused space to useful storage ratio for single relation per chip/segment per chip/segment distributed storage organization.

$$d = m \exp(m/D_A)/(\exp(m/D_A)-1) - \exp(1/D_A)/(\exp(1/D_A)-1)$$

and

$$t = SL \exp(SL/T_A)/(\exp(SL/T_A)-1) - \exp(1/T_A)/(\exp(1/T_A)-1)$$

From the results of our discussion on fragmentation in bubble chips, it is clear that storage organization I is the best choice among the three organizations if one wants to minimize unused storage.

5.4. Summary on Storage Organizations

In Table 7, several characteristics of the three storage organizations are summarized. Organization I has the smallest sort-merge time complexity, achieves the best storage utilization and requires the least amount of extra storage of all three organizations. However, it allows only 1/S of a data base to be accessed concurrently. Organizations II and III allow all relations in a data base to be accessed concurrently. (We note here that all three organizations have equivalent throughputs, which can be obtained by multiplying the degree of concurrency by the efficiency of operations.) The least amount of control complexity in terms of number and size of comparators is achieved in organization II, at the cost of longer comparison times in the control units. The choice of organization III for a data base can only be justified by its degree of concurrency and for very large relations (number of tuples in the order of

Table 7. Summary on Three Storage Organizations.

	Organization I	Organization II	Organization III
Extra Storage	CB/SL	CB/S	CB/S
Degree of Concurrency	1/S	1	1
Comparators	C/W W-bit 2C/Mm (2Mm/W)-bit	C bit serial 2C/Mm bit serial	C/W W-bit 2C/Mm (2Mm/W)-bit
Bubble Chip Organization	Major-minor Loop	Major-minor Loop Bubble Ladder	Major-minor Loop
Unused/Useful Storage Ratio (Uniform Distrib.)	$\frac{m}{2D_A} + \frac{L}{2T_A} + \frac{mL}{4D_AT_A}$	$\frac{mL}{2D_A} + \frac{S}{2T_A} + \frac{mLS}{4D_AT_A}$	$\frac{m}{2D_A} + \frac{SL}{2T_A} + \frac{mLS}{4D_AT_A}$
Sort-merge Complexity	O(TL)	O(TSL)	O(TSL)

chip size, B, in order to get an efficient storage utilization).

A possible compromise in the choice of storage organizations for a data base is the use of organization I to store relations that are frequently requested, and the use of organization II to store relations that are seldomly requested. Processing of queries involving frequently requested relations, once initiated, could be performed relatively fast, while queries involving seldomly requested relations would be processed autonomously at a much slower pace.

5.5. Algorithms for Relational Algebraic Operations

We consider here algorithms for relational algebraic operations (projection, join, selction, division, intersection, union, difference, and Cartesian product). We note that in performing these operations, tuples have to be read out in order to be compared. In organizations I and III, data rearrangement operations take one bubble shift and no additional bubble shifts are required for comparison of tuples. On the other hand, in organization II, either data rearrangement operations (if the uniform bubble ladder chip organization is used) or comparisons (if the major-minor loop chip organization is used) require L bubble shifts. Therefore, the results on the performance of algorithms (in terms of number of bubble shifts) in organization II with uniform bubble ladder can be obtained by multiplying the results for organization I by L. When the major-minor loop chip organization is used in organization II, the dominant term

in the time complexity of the algorithms is the number of shifts required in the comparison of tuples. In this case, when comparison of tuples is not needed, the mark bits used to carry out the primitive relational algebraic operations make it unnecessary to read out an entire tuple.

Projection is the operation chosen to illustrate the detailed implementation of relational algebraic operations in organizations I and II. Informal descriptions of algorithms and worst case time complexity results are given for all other relational algebraic operations. Organization III is not discussed here.

5.5.1. Projection(Relation,Domainset)

Two mark bits per tuple are needed to store information about tuples being examined. Initially, they are set to 00 and the meaning of their values is as follows: 00 - tuple not examined or not a duplicate tuple, 01 - duplicate tuple, and 11 - candidate or result tuple.

Projection is performed in two phases. During phase 1, performed simultaneously in all super modules containing Relation, tuples in each super module are compared, on Domainset, against each other. The mark bits of duplicate tuples are set to 01, the remaining tuples have their mark bits set to 11. During phase 2, candidate tuples, the ones marked 11 during phase 1, are compared across super modules to eliminate duplicate tuples whose mark bits are then set to 01. The tuples marked

11 at the end of phase 2 constitute the result relation of the projection of Relation on Domainset. Algorithms for projection in organization I are presented in Appendix C. We assume that all tuples of relations on which operations are being applied are already in the major loops or in auxiliary storage.

Algorithm C.1, used in phase 1, is very similar to the ones based on the bubble sort, described in Chapter 4. Alternate passes through tuples stored in the major loops are done in opposite shift directions. Four $\log_2 L$-bit counters are needed. The number of passes in Algorithm C.1, for a particular super module, is equal to the number of distinct tuples in the super module plus one. However, in the worst case, all tuples in a super module distinct, the number of passes is equal to the number of distinct tuples, L, and the worst case time complexity is equal to $L(L+1)/2$ bubble shifts.

Two algorithms are possible for phase 2: (1) broadcasting of candidate tuples from one super module to other super modules, and (2) comparison between candidate tuples from pairs of super modules. In Algorithm C.2, the number of tuples that are broadcast can be minimized by choosing, in step 1, the super module with the smallest number of candidate tuples. Registers for this purpose, holding the number of candidate tuples in each super module, can also be used to avoid unnecessary bubble shifts. The result relation can be output simultaneously with tuple broadcasting since all broadcast tuples are result tuples. The worst case time complexity of Algorithm C.2 is $(\lceil T/L \rceil -1)(L+1)L$

bubble shifts.

In Algorithm C.3, pairs of super modules are processed simultaneously. Again, unnecessary bubble shifts can be avoided by the use of registers holding the number of candidate tuples in each super module. The number of iterations in Algorithm C.3 is equal to the number of pairings of super modules. Hence, the maximum amount of parallelism is achieved when the number of pairings is minimum, that is, when it is equal to $\lceil T/L \rceil - 1$, if $\lceil T/L \rceil$ is even, and when it is equal to $\lceil T/L \rceil$, if $\lceil T/L \rceil$ is odd. (For example, with 5 super modules numbered from 1 to 5 we have the following pairings: (1,2), (3,4); (1,3), (4,5); (1,4), (2,5); (1,5), (2,3); and (2,4), (3,5). With 4 super modules the pairings are: (1,2), (3,4); (1,4), (2,3); and (2,4), (3,5).) Although the pairings for maximum parallelism are fixed, the order in which they are chosen is irrelevant. Step 2 of Algorithm C.3 is similar to Algorithm C.1 and is performed in each super module of a pair of super modules. One candidate tuple from each super module of a pair of super modules is sent to the other super module and comparisons are done simultaneously in both super modules. Each pass of comparisons is followed by a reversal of shift direction and by another pass of comparisons. This procedure is repeated until all candidate tuples in both super modules are compared. Step 2 requires, at most $L(L+1)/2$ bubble shifts.

Unlike Algorithm C.2, tuples of the result relation can be output only after the end of Algorithm C.3, which means

an addition, at most, of L bubble shifts. Therefore, including bubble shifts to read out the result relation, the worst case time complexity of Algorithm C.3 is $L + (\lceil T/L \rceil - 1)(L+1)L/2$ bubble shifts, if $\lceil T/L \rceil$ is even, and $L + (\lceil T/L \rceil)(L+1)L/2$ bubble shifts, if $\lceil T/L \rceil$ is odd.

Combining the results of the efficiencies of algorithms in phases 1 and 2, we arrive at the worst case time complexities of $(\lceil T/L \rceil)L^2 + (\lceil T/L \rceil)L - L^2/2 - L/2$ bubble shifts, for Algorithms C.1 and C.2, and, $(\lceil T/L \rceil)L^2/2 + (\lceil T/L \rceil)L/2 + L$ bubble shifts, if $\lceil T/L \rceil$ is even, $(\lceil T/L \rceil)L^2/2 + (\lceil T/L \rceil)L/2 + L^2/2 + 3L/2$ bubble shifts, if $\lceil T/L \rceil$ is odd, for Algorithms C.1 and C.3. These figures can be reduced by overlapping phase 1 with the first iteration of phase 2, a savings of roughly half a broadcasting iteration and one pairing iteration for Algorithms C.2 and C.3, respectively. However, these savings can be achieved only at the cost of doubling the total number of comparators.

Improvement on the efficiency of the projection operation can also be attained with the use of a working space, an additional number of super modules capable of storing tuples in the same way as the regular storage. Algorithm C.4, used in this approach, requires at most $2(\lceil T/L \rceil - 1)$ super modules of working space. Candidate tuples are transferred to the working space simultaneously with the execution of Algorithm C.1. This step in Algorithm C.4 is repeated with the super modules containing candidate tuples from the previous iteration for as long as the

decrease in the number of super modules containing candidate tuples warrants its application, or until there is only one super module containing candidate tuples left. This iterative procedure is followed by the application of Algorithms C.1 and C.2 or C.3 to the remaining super modules. The worst case time complexity of Algorithm C.4 is worse than that for the algorithms without a working space. However, substancial savings on the number of bubble shifts required can be achieved, depending on the amount of decrease in the number of super modules which contain candidate tuples after step 1 of Algorithm C.4.

The performance of the algorithms in this section is dependent not only on the number of tuples in the result relation but also on the original distribution of the tuples of Relation among the super modules. Although Algorithm C.2 is more efficient than Algorithm C.3, it requires a more complex control because of the determination and bookkeeping of super module pairings.

The algorithms for projection in storage organization II are essentially the same as the ones for projection in organization I. The same results for time complexity of algorithms, now in terms of number of data rearrangement operations (uniform bubble ladder) or number of comparisons, with the replacement of L by S in the expressions obtained, are valid. As pointed out before, either data rearrangement operations or comparisons in organization II require L bubble shifts. Therefore, the time complexities for the algorithms for projection in organization II are $O(TLS)$ bubble shifts for Algorithms C.1 and

C.2, and O(TLS/2) bubble shifts for Algorithms C.1 and C.3.

5.5.2. Join(Relation1,Domain1,Condition,Relation2,Domain2)

Two approaches are possible: (1) output result relation as the join is being performed into temporary storage, and (2) preprocess relations by making use of mark bits in the tuples and examine only marked tuples at output time to obtain the result relation. We note that, in the second approach, marked tuples are examined twice, once in the preprocessing of tuples and once at output time. Both approaches require the examination of tuples from all pairs of super modules, each pair constituted of one super module from each relation. Each tuple in a super module is compared with all tuples in the other super module to check if Condition is satisfied. Pairs of super modules can be processed in parallel and L^2 data rearrangement operations are required in organization I, S^2 data rearrangement operations and comparisons in organization II. Therefore, if the two relations have T_1 and T_2 tuples, $\max(T_1/L , T_2/L) L^2$ data rearrangement operations in organization I and $\max(\lceil T_1/S \rceil , \lceil T_2/S \rceil) S^2$ data rearrangement operations and comparisons in organization II, are required to join them. Hence, the time complexity of the join operation is $O(\max(T_1,T_2)L)$ bubble shifts in organization I and $O(\max(T_1,T_2)SL)$ bubble shifts in organization II.

5.5.3. Selection(Relation,Condition)

All tuples in Relation are examined and the ones satis-

fying Condition are marked or output. Super modules are processed in parallel. The time complexity of the selection operation is O(L) bubble shifts in organization I and O(SL) bubble shifts in organization II.

5.5.4. Union(Relation1,Relation2)

Relations are processed in a way similar to the processing of relations in the join operation. Instead of a single domain value the entire tuples are compared and the Condition to be satisfied by tuples is not equal (to eliminate duplicates). Once the tuples of the result relation are marked, there is no need for further comparisons. The union operation has the same time complexity of the join operation.

5.5.5. Intersection(Relation1,Relation2)

Processing of relations is similar to the processing in the union operation, except that the Condition to be satisfied by tuples is equal and tuples in one relation only need to be marked. Again, time complexity is the same as for the join operation.

5.5.6. Difference(Relation1,Relation2)

Again, processing of relations is similar to the processing in the union operation, except that only tuples in Relation1 are marked or output. Difference has the same time complexity as the join operation.

5.5.7. Cartesian product(Relation1,Relation2)

Output concatenation of tuples in Relation1 with tuples in Relation2. If no output order is required than $\min(\lceil T_1/S \rceil, \lceil T_2/S \rceil)$ (or $\min(\lceil T_1/L \rceil, \lceil T_2/L \rceil)$) tuples can be read out at a time but only one tuple, otherwise. However, the time complexity for Cartesian product is the same as for the join operation since all these tuples are available for output at the same time.

5.5.8. Division(Relation1,Domainset1,Domainset2,Relation2)

The description that follows applies to organizations I and II. However, the intermediate expressions of time complexities are given for organization II with major-minor loop chip organization. The same considerations given for the projection operation can be applied to obtain expressions for organization I that lead to the final expressions of time complexity for organization I.

One mark bit is needed for each tuple in Relation1. All mark bits are initially set to 0. First, we project Relation2 on Domainset2 and store the number of tuples obtained in a register. At most, $\lceil T_2/S \rceil S^2$ comparisons are required in this projection operation. Tuples in Relation1 are then examined against each tuple in the projection, for occurrence of tuples in the projection, in Domainset1 of tuples in Relation1. Mark bits of tuples in Relation1 are set to 1 when such occurrence happens. Tuples already marked 1 are not examined twice. This

step takes at most $\lceil T_1/S \rceil S^2$ comparisons.

In the next step, we examine marked tuples in Relation1 for duplicates on the set of domains not in Domainset1, increasing the value of a counter at each occurrence of a duplicate and resetting the correspondent mark bit to 0. If, during this process, counter becomes equal to the contents of the register, the mark bit of the last duplicate is not reset. This step requires at most $\lceil T_1/S \rceil S^2$ comparisons. The result relation is the one obtained by projecting the relation constituted by all tuples marked 1 in Relation1, on the set of domains not in Domainset1. The worst case time complexity of the division operation can be derived as being $O((2T_1 + T_2)L)$ bubble shifts in organization I and $O((2T_1 + T_2)LS)$ bubble shifts in organization II.

6. CONCLUSION

In this thesis, we explored ways of incorporating the novel bubble chip organizations and bubble movement operations in the design of intelligent memories. Specific results on the performance of file processing algorithms, basic relational algebraic operations, as well as memory organizations and hierarchical memory systems are presented in the previous chapters and will not be repeated here.

These results can be used in the design of a data base machine supporting the relational data model. In particular, the logical step to follow the study in this thesis is the integration of the several algorithms into the design of the memory interface, control units, primitive operations control processor (POCP), and the interconnections among themselves. Such a design should be carried out to a level elementary enough to permit the determination of realistic estimates for hardware cost in the intelligent memory.

We conclude this thesis by listing some topics and areas of further research:

(1) Fragmentation (unused space) in bubble chips and its effect on data retrieval performance of memory organizations and on the performance of the several algorithms.

(2) Design and evaluation of algorithms, other than simple sorting, for clustering in magnetic bubble memories.

(3) Design of control mechanisms and modifications in

the algorithms to allow concurrency of execution of two or more operations on the same relation.

(4) Detailed analysis to determine average time complexities of algorithms for primitive relational algebraic operations.

(5) Simulation studies to estimate the time complexities of the several algorithms under various assumptions for data base contents and configuration.

LIST OF REFERENCES

[1] Slotnick, D. L., "Logic per Track Devices," in Advances in Computers, vol. 10, Franz Alt., Ed., Academic Press, New York, 1970, pp. 291-296.

[2] Healy, L. O., K. L. Doty, G. J. Lipovski, "The Architecture of a Context Addressed Segment Sequential Storage," Proc. AFIPS 1972 FJCC, vol. 41, Pt. II, pp. 691-701.

[3] Parhami, B., "A Highly Parallel Computer System for Information Retrieval," Proc. AFIPS 1972 FJCC, vol. 41, pp. 681-690.

[4] Parker, J. L., "A Logic per Track Retrieval System," Proc. IFIP Congress, 1971, pp. 711-716.

[5] Minsky, N., "Rotating Storage Devices as Partially Associative Memories," Proc. AFIPS 1972 FJCC, vol. 41, pp. 587-595.

[6] Copeland, G. P.,G. J. Lipovski, S. Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-Numeric Processing," Proc. of the First Annual Symposium on Computer Architecture, 1973, pp. 121-128.

[7] Su, S. Y. W. and G. J. Lipovski, "CASSM: A Cellular System for Very Large Data Bases," Proc. Int. Conf. on Very Large Data Bases, 1975, pp. 456-472.

[8] Ozkarahan, E. A., S. A. Schuster, K. C. Smith, "RAP -- An Associative Processor for Data Base Management," Proc. AFIPS 1975 NCC, vol. 44, pp. 53-65.

[9] Lin, C. S., D. C. P. Smith, J. M. Smith, " The Design of a Rotating Associative Memory for Relational Database Applications," ACM Trans. Database Systems, vol. 1, No. 1, 1976, pp. 53-65.

[10] Hsiao, D. K. and K. Kannan, "The Architecture of a Data Base Computer," Technical Reports OSU-CISRC-TR-76-1, 2, and 3. Computer and Information Science Research Center, Ohio State University, Columbus, Ohio.

[11] Edelberg, M. and L. R. Schissler, "Intelligent Memory," Proc. ACM 1976 NCC, pp. 393-400.

[12] Smith, J. M. and P. Y. Chang, "Optimizing the Performance of a Relational Algebra Data Base Interface," Comm. ACM, vol. 18, No. 10, 1975, pp. 568-579.

[13] Bobeck, A. H., P. I. Bonyhard, J. E. Geusic, "Magnetic Bubbles -- An Emerging New Memory Technology," Proc. of the IEEE, vol. 63, No. 8, 1975, pp. 1176-1195.

[14] Chang, H., "Magnetic Bubble Technology -- Present and Future," Proc. 7th Conf. on Solid State Devices, Supp. to Japanese J. of Applied Physics, vol. 15, pp. 5-10, 1976.

[15] Pohm, A. V., et al., "Bubble Memories for Microcomputers and Minicomputers," IEEE Trans. on Magnetics, vol. 12, No. 6, Nov. 1976, pp. 636-638.

[16] Beausoleil, W., D. T. Brown, B. E. Phelps, "Magnetic Bubble Memory Organization," IBM J. Research and Dev., 1972, pp. 587-591.

[17] Bonyhard, P. I. and T. J. Nelson, "Dynamic Data Reallocation in Bubble Memories," The Bell System Technical Journal, vol. 52, No. 3, 1973, pp. 307-317.

[18] Wong, C. K. and P. C. Yue, "Data Organization in Magnetic Bubble Lattice File," IBM J. Research and Dev., 1976, pp. 576-581.

[19] Wong, C. K. and D. Coppersmith, "The Generation of Permutations in Magnetic Bubble Memories," IEEE Trans. on Computers, vol. C-25, No. 3, 1976, pp. 254-262.

[20] Tung, C., T. C. Chen, H. Chang, "Bubble Ladder for Information Processing," IEEE Trans. on Magnetics, vol. Mag-11, No. 5, 1975, pp. 1163-1165.

[21] Chen, T. C. and C. Tung, "Storage Management Operations in Linked Uniform Shift-Register Loops," IBM J. Research and Dev., 1976, pp. 123-131.

also
Chen, T. C., et al., "Simplified Odd-Even Sort Using Multiple Shift-Register Loops," IBM Research Report, also to appear in Int. J. Computer and Information Science, 1978.

[22] Kluge, W. E., "Data File Management in Shift Register Memories," International Conference on Very Large Data Bases, 1977.

[23] Chang, H., J. Fox, D. Lu, L. L. Rosier, "A Self-Contained Magnetic Bubble-Domain Memory Chip," IEEE Trans. on Magnetics, 1972, pp. 214-222.

[24] Chang, H. "Bubble Domain Memory Chips," IEEE Trans. on Magnetics, vol. 8, No. 3, pp. 564-569, Sept. 1972.

[25] Calhoun, B. A., et al., "Column-Access of a Bubble Lattice: Column Translation and Lattice Translation," IBM J. Research and Dev., vol. 10, pp. 368-375, 1975.

[26] Ho, G. P. and H. Chang, "Field Access Bubble Lattice Propagation Devices," IEEE Trans. on Magnetics, vol. Mag-13, pp. 945-952, 1977.

[27] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Comm. ACM, vol. 13, No. 6, pp. 377-387, June 1970.

[28] Codd, E. F., "Further Normalization of the Data Base Relational Model," in Data Base Systems, Courant Computer Science Symposia Series, vol. 6, Englewood Cliffs, N. J.: Prentice-Hall, 1972, pp. 33-61.

[29] Codd, E. F., "Relational Completeness of Data Base Sublanguages," in Data Base Systems, Courant Computer Science Symposia Series, vol. 6, Englewood Cliffs, N. J.: Prentice-Hall, 1972, pp. 65-98.

[30] Saltzer, J. H., "A Simple Linear Model of Demand Paging Performance," Comm. ACM, vol. 17, No. 4, pp. 181-185, 1974.

[31] Kuck, D. J. and D. H. Lawrie, "The Use and Performance of Memory Hierarchies: A Survey," Software Engineering, vol. 1, 1970, pp. 45-77.

[32] Baskett, F. and A. Rafii, "The Aϕ Inversion Model of Program Paging Behavior," Technical Report STAN-CS-76-579, Computer Science Department, Stanford University, 1976.

[33] Ragaz, N. and J. Rodrigues-Rosell, "Empirical Studies of Storage Management in a Data Base System," IBM Technical Report RJ 1834(26703), 1976.

[34] Knuth, D. E., The Art of Computer Programming, vol. 3, Sorting and Searching, Addison-Wesley, 1973, pp. 105-111.

[35] Knuth, D. E., The Art of Computer Programming, vol. 3, Sorting and Searching, Addison-Wesley, 1973, pp. 357-361.

[36] Hollaar, L. A., "A List Merging Processor for Inverted File Information Retrieval Systems," Report 762, Department of Computer Science, University of Illinois, Oct. 1975.

Appendix A: RETRIEVAL TIMES IN BUBBLE MEMORY ORGANIZATIONS

Average retrieval times in the three possible ways of distributing words and bits of a page among chips, page per chip, word per chip and bit per chip organizations, described in Section 3.1, are discussed here. We consider the possible alternatives of organizing words and bits in a bubble chip for the three basic bubble chip organizations, major-minor loop, on-chip decoder and bubble lattice organizations. The capacity of a chip is B bits (L S-bit storage loops).

A.1. Page per Chip Organizations

In this case, all words of a given page are stored in the same chip. We consider the three bubble chip organizations separately.

A.1.1. Major-minor Loop Chip Organization

There are two possible ways of organizing words inside a chip: (1) word per minor loop and (2) word distributed among minor loops.

To access a word stored entirely in a minor loop, three steps are involved: (1) move word to the first or the Lth minor loop (the closest one), (2) move word inside the first or the Lth minor loop to position it to be transferred to the major loop, and (3) read out word, bit by bit. Two bubble shifts are necessary to move one bit from one minor loop to a neighboring

minor loop, one shift in the major loop and another one in the minor loop. Therefore, on the average, steps 1, 2 and 3 require B/2, S/4 and 2W shifts, respectively. Furthermore, to move data back to their original positions, these steps are applied in the reverse order, with the shift direction also reversed, doubling the number of shifts required to access a word. If word size, W, is equal to the minor loop length, S, step 2 is not needed. Hence, the average word retrieval times, expressed as the number of shifts per bit retrieved, are

$T_w = 4 + B/W + S/2W$ (*) and

$2 + B/2W + S/4W$, if $S > W$,

and

$T_w = 4 + B/W$ (*) and

$2 + B/2W$, if $S = W$,

where (*) include shifts to move data back to their original positions.

Steps similar to the above, with respect to a page, are needed for page retrieval. On the average, the corresponding steps 1, 2 and 3 require B/2, S/4 and 2NW shifts, respectively. Step 2 is not needed if $S \leq NW$. Moving a page back to its original position requires the same number of shifts required for its access. Hence, the average page retrieval times are

$T_p = 4 + B/NW + S/2NW$ (*) and

$T_p = 2 + B/2NW + S/4NW$, if $S > NW$,

and

$T_p = 4 + B/NW$ (*) and

$2 + B/2NW$, if $S \leq NW$,

where (*) include shifts to move data back to their original positions.

A more efficient way to organize words in a chip is to have each word from a page distributed among minor loops. In this case, a word is brought bit parallelly to the major loop and then shifted inside the major loop to be read out. To access a word in this organization, the following steps are involved: (1) bring word, in parallel, to the major loop, (2) move word inside major loop to read/write position and (3) read out word, bit by bit. On the average, steps 1, 2 and 3 require $S/4 + 1$, $B/4S$ and W shifts, respectively. Again, to move a word back to its original position, the number of shifts to access a word is doubled. However, if word size, W, is equal to the number of minor loops, L (= B/S), step 2 is not needed. Hence, the average word retrieval times are

$T_w = 2 + 2/W + B/2SW + S/2W$ (*) and

$1 + 1/W + B/4SW + S/4W$, if $B/S > W$,

and

$T_w = 2 + 2/W + S/2W$ (*) and

$1 + 1/W + S/4W$, if $B/S = W$,

where (*) include shifts to move data back to original positions.

Steps similar to the above, with respect to a page, are needed for page retrieval. On the average, the corresponding steps 1, 2 and 3 require $S/4 + 1$, $B/4S$ and NW shifts, respectively. If $B/S \leq NW$, step 2 is not needed and step 1, on the average, requires $S/4 + SNW/B$ shifts. Hence, the average page retrieval times are

$$T_p = 2 + 2/NW + B/2SNW + S/2NW \quad (*) \quad \text{and}$$

$$1 + 1/NW + B/2SNW + S/4NW, \quad \text{if} \quad B/S > NW,$$

and

$$T_p = 2 + 2S/B + S/2NW \quad (*) \quad \text{and}$$

$$1 + S/B + S/4NW, \quad \text{if} \quad B/S \leq NW,$$

where (*) include shifts to move data back to original positions.

A.1.2. On-chip Decoder Chip Organization

Again, two ways of organizing words inside a chip are possible: (1) word per storage loop and (2) word distributed among storage loops.

To access a word stored entirely in a storage loop, three steps are involved: (1) move word to the input of the decoder, (2) shift word inside the decoder and (3) read out word, bit by bit. To move a bit from the input to the output of the decoder, $\log_2 B/S$ ($\log_2 L$) shifts are necessary. Therefore, on the average, steps 1, 2 and 3 require $S/4$, $\log_2 B/S$ and W shifts,

respectively. If $S = W$, there is no need for step 1 and only $\log_2 B/S$ additional shifts are necessary to move a word back to its original position. Hence, the average word retrieval times are

$$T_w = 2 + \frac{2}{W}\log_2 B/S + S/2W \quad (*) \quad \text{and}$$

$$1 + \frac{1}{W}\log_2 B/S + S/4W, \quad \text{if} \quad S > W,$$

and

$$T_w = 1 + \frac{2}{W}\log_2 B/S \quad (*) \quad \text{and}$$

$$1 + \frac{1}{W}\log_2 B/S, \quad \text{if} \quad S = W,$$

where (*) include shifts to move data back to their original positions.

Steps similar to the above, with respect to a page, are needed for page retrieval. On the average, the corresponding steps 1 and 3 require $S/4$ and NW shifts, respectively. However, for step 2, the number of shifts for the access of an entire page is $\log_2 B/S$, if the stepped control of address lines is used, and $K\log_2 B/S$ shifts (where K is the number of storage loops involved in the page access), otherwise. If $S \leq NW$, NW/S storage loops are involved and step 1 is not needed. Furthermore, step 3 does not have to be reversed to move a page back to its original position. Hence, the average page retrieval times are

$$T_p = 2 + \frac{2}{NW}\log_2 B/S + S/2NW \quad (*) \quad \text{and}$$

$$T_p = 1 + \frac{1}{NW}\log_2 B/S + S/4NW, \quad \text{if} \quad S > NW,$$

and

$$T_p = 1 + \frac{2}{S}\log_2 B/S \quad (*),$$

$$1 + \frac{1}{S}\log_2 B/S,$$

$$1 + \frac{2}{NW}\log_2 B/S \quad (*)\ (+) \quad \text{and}$$

$$1 + \frac{1}{NW}\log_2 B/S \quad (+), \quad \text{if} \quad S \leq NW,$$

where (+) use stepped control of address lines and (*) include shifts to move data back to their original positions.

Alternatively, we can have a word from a page distributed among storage loops in a chip. In this case, a word is brought bit parallelly to the input of the decoder. Again, three steps are involved in accessing a word: (1) move word bit parallelly to the input of the decoder, (2) shift word inside the decoder, and (3) read out word, bit by bit. On the average, steps 1, 2 and 3 require $S/4$, $W\log_2 B/S$ and W shifts, respectively. We note that step 2 always requires $W\log_2 B/S$ shifts regardless of the relationship between W and B/S. Hence, the average word retrieval times are

$$T_w = 2 + 2\log_2 B/S + S/2W \quad (*),$$

$$1 + \log_2 B/S + S/4W,$$

$$2 + \frac{2}{W}\log_2 B/S + S/2W \quad (*)\ (+) \quad \text{and}$$

$$1 + \frac{1}{W}\log_2 B/S + S/4W \quad (+),$$

where (+) use stepped control of address lines and (*) include shifts to move data back to their original positions.

Steps similar to the above, with respect to a page, are needed for page retrieval. On the average, the corresponding steps 1, 2 and 3 require S/4, $W\log_2 B/S$ and NW shifts, respectively. Hence, the average page retrieval times are

$$T_p = 2 + 2\log_2 B/S + S/2NW \quad (*),$$

$$1 + \log_2 B/S + S/4NW,$$

$$2 + \frac{2}{NW}\log_2 B/S + S/2NW \quad (*)\ (+) \quad \text{and}$$

$$1 + \frac{1}{NW}\log_2 B/S + S/4NW \quad (+),$$

where (+) use stepped control of address lines and (*) include shifts to move data back to their original positions.

A.1.3. Bubble Lattice Chip Organization

There are two possible ways of organizing words inside a chip: (1) word per column and (2) word distributed among columns.

To access a word stored entirely in a column (not in buffer zones), three steps are involved: (1) move word to the access channel, bit parallelly, (2) move word inside the access channel to the closest read/write position and (3) read out word, bit by bit. On the average, steps 1, 2 and 3 require

B/2S + 1/2, S/4 and W shifts, respectively. To move a word back to its original position, the number of shifts to access a word is doubled. However, if word size, W, is equal to column length, S, step 2 is not needed. Furthermore, immediately after reading out a word, it is in the position required, inside the access channel, to put it back into the columns. Hence, the average word retrieval times are

$T_w = 2 + 1/W + B/SW + S/2W$ (*) and

$1 + 1/2W + B/2SW + S/4W$, if $S > W$

and

$T_w = 1 + 1/W + B/SW$ (*) and

$1 + 1/2W + B/2SW$, if $S = W$,

where (*) include shifts to move data back to their original positions.

Steps similar to the above, with respect to a page, are needed for page retrieval. On the average, the corresponding steps 1, 2 and 3 require B/2S + 1/2, S/4 and NW shifts, respectively. If $S \leq NW$, step 2 is not needed and step 1, on the average, requires $(S/2 + S^2)/NW$ shifts. Furthermore, immediately after reading out all words in the access channel, they are in the positions required to put them back into the columns. Hence, the average page retrieval times are

$T_p = 2 + 1/NW + B/SNW + S/2NW$ (*) and

$1 + 1/2NW + B/2SNW + S/4NW$, if $S > NW$,

and

$$T_p = B/SNW + (S + S^2)/(NW)^2 \quad (*) \quad \text{and}$$

$$B/2SNW + (S/2 + S^2)/(NW)^2, \quad \text{if} \quad S \leq NW,$$

where (*) include shifts to move data back to their original positions.

Alternatively, we can have a word distributed among the columns (not in buffer zones) of a chip. To access a word in this organization, three steps are involved: (1) move the bits of the word to the access channel, (2) shift bits of the word inside the access channel to the closest read/write position, and (3) read out word, bit by bit. On the average, steps 1, 2 and 3 require $B/2S + W/2$, $SW/4$ and W shifts, respectively. If $B/S = W$, step 1 requires only W shifts. To move a word back to its original position, the number of shifts is doubled. Hence, the average word retrieval times are

$$T_w = 3 + B/SW + S/2 \quad (*) \quad \text{and}$$

$$3/2 + B/2SW + S/4, \quad \text{if} \quad B/S > W,$$

and

$$T_w = 4 + S/2 \quad (*) \quad \text{and}$$

$$2 + S/4, \quad \text{if} \quad B/S = W,$$

where (*) include shifts to move data back to their original positions.

Steps similar to the above, with respect to a page, are

needed for page retrieval. On the average, the corresponding steps 1, 2 and 3 require B/2S + NW/2, SNW/4 and NW shifts, respectively. Again, if B/S $\leq$ NW, step 1 requires only NW shifts. Hence, the average page retrieval times are

T_p = 3 + B/SNW + S/2 (*) and

3/2 + B/2SNW + S/4, if B/S > NW,

and

T_p = 4 + S/2 (*) and

2 + S/4, if B/S $\leq$ NW,

where (*) include shifts to move data back to their original positions.

A.2. Word per Chip Organizations

In this case, each chip contains only one word from any given page. It is not difficult to see that average word retrieval times for page per chip and word per chip organizations have identical expressions for the corresponding word organizations (word per loop and word distributed), in the three basic bubble chip organizations.

On the other hand, a page in the word per chip organization is accessed word parallelly. The average number of shifts to retrieve a page is identical to the average number of shifts to retrieve a word. Therefore, as shown in Table 3 of Section 3.2, average page retrieval times in word per chip organizations

are obtained by simply dividing the average word retrieval times by the number of words in a page, N.

A.3. Bit per Chip Sequential Memory Organizations

In this case, each word has only one bit in each chip and each page has up to S (or L) bits in each chip. All steps in the access of a word are bit parallel. The derivation of average word retrieval times in this organization is straightforward. A simple computation of the average number of shifts to read out one bit from a chip followed by the division of the expressions obtained by W yields the results tabulated in Table 4 of Section 3.2. The derivation of average page retrieval times for the three bubble chip organizations is given here.

A.3.1. Major-minor Loop Chip Organization

There are two possible ways of organizing bits of a page inside a chip: (1) page per minor loop, and (2) page distributed among minor loops.

Steps similar to the ones in the page per chip/word per loop organization are required to access a page stored in a minor loop. On the average, the corresponding steps 1, 2 and 3 require B/2, S/4 and 2N shifts, respectively. However, if $S \leq N$, additional chips are used to store the page and, therefore, N/S words of the page are accessed in parallel. Also, step 2 is not needed and step 3 requires 2S shifts. Hence, the average page retrieval times are

$T_p = 4/W + B/NW + S/2NW$ (*) and

$2/W + B/2NW + S/4NW$, if $S > N$,

and

$T_p = BS/N^2W + 4S^2/N^2W$ (*) and

$BS/2N^2W + 2S^2/N^2W$, if $S \leq N$,

where (*) include shifts to move data back to their original positions.

Analogously, steps similar to ones in page per chip/word distributed organization are required to access a page distributed among minor loops. On the average, the corresponding steps 1, 2 and 3 require $S/4 + 1$, $B/4S$ and N shifts, respectively. However, if $B/S \leq N$, additional chips are used to store the page and, therefore, NS/B words of the page are accessed in parallel. Also, step 2 is not needed and step 3 requires B/S shifts. Hence, the average page retrieval times are

$T_p = 2/W + 2/NW + B/2SNW + S/2NW$ (*) and

$1/W + 1/NW + B/4SNW + S/4NW$, if $B/S > N$,

and

$T_p = 2B/SN^2W + B/2N^2W + 2B^2/S^2N^2W$ (*) and

$B/SN^2W + B/4N^2W + B^2/S^2N^2W$, if $B/S \leq N$,

where (*) include shifts to move data back to their original positions.

A.3.2. On-chip Decoder Organization

The discussion on the derivation of expressions for average page retrieval times in the on-chip decoder chip organization would be analogous to the one above and will not be presented here. The expressions for average page retrieval times are listed in Table 4 (Section 3.2).

A.3.3. Bubble Lattice Chip Organization

Again, the discussion on the derivation of expressions for average page retrieval times in the bubble lattice chip organization will not be presented for the same reason given above. The expressions for average page retrieval times can also be found in Table 4 (Section 3.2).

A.4. Bit per Chip Random Memory Organization

In this case, we also use W chips to store a word from a page, but NW chips are used to store a page. Therefore, a chip may contain bits from up to B pages. The derivations of average word and page retrieval times are straightforward and the expressions for retrieval times are given in Table 5 (Section 3.2).

Appendix B: SORTING ALGORITHMS

After execution of the sorting algorithms, items are arranged in non-decreasing order of their key values, starting from location 0 in the data loop.

Algorithm B.1: Basic sort in the first model

```
begin C = N; P = 0; D = "1"; /* initialize counters, register */
      apply (a); /* get first item */
S1:   C1 = C = C - 1; P = P + 1; /* update counters */
      if C = 1 then goto S3; /* test end of sort */
L1:  /* pass seeking smallest key */
      if  A < B  then begin  D = "0";
                             apply (cd); /* exchange A, B */
                      end;
      if  C1 > 1  then  apply (a); /* get next item */
      C1 = C1 - 1;
      if  C1 > 0   then goto  L1; /* test end of pass */
      if  D = "1"  then goto  S3; /* test end of sort */
      apply (caa); /* prepare items for next pass */
      D = "1"; /* set register for next pass */
S2:   C1 = C = C - 1; P = P + 1; /* update counters */
L2:  /* pass seeking largest key */
      if  A > B  then begin D = "0";
```

```
                        apply (cd); /* exchange A, B */
                    end;
        if  C1 > 1  then  apply (a); /* get next item */
        C1 = C1 - 1;
        if  C1 > 0  then goto  L2; /* test end of pass */
        /* test end of sort */
        if  D = "0"  then
                    begin D = "1"; /* set register */
                          apply (caa); /* prepare items */
                          goto  S1   /* next pass */
                    end;
S3:  /* move ordered items to final locations in data loop */
        if  (P odd)  then
                    begin apply (ac);
                          apply (a)  (N-(P-3)/2) mod N times
                    end
                    else  apply  (a)  P/2  times
end  /* of Algorithm B.1  */
```

Algorithm B.2: Basic sort in the second model

```
begin C = N; P = 0; D = "1"; /* initialize counters, register */
        apply (a); /* get first item */
S1:     C1 = C = C - 1; P = P + 1; /* update counters */
        if  C = 1  then goto  S2; /* test end of sort */
```

```
L:  /* pass seeking largest key */
      if  A > B  then
                 begin  D = "0";
                        apply  (ea); /* exchange A, B */
                 end;
      apply  (a); /* get next item */
      C1 = C1 - 1;
      if  C1 > 0  then goto  L; /* test end of pass */
      if  D = "0"  then
                 begin D = "1"; /* set detector */
                       apply (a)  P times; /* prepare items */
                       goto  S1  /* iterate pass */
                 end;
S2:  /* move ordered items to final locations in data loop */
      apply  (a)  (P - 1)  times;
end   /* of  Algorithm B.2  */
```

Algorithm B.3: Basic sort in the third model

```
begin  C = N;  P = 0; D = "1"; /* initialize counters, register*/
       apply  (a);  /* get first item */
S1:    C1 = C = C - 1; P = P + 1; /* update counters */
       if  C = 1  then goto  S3; /* test end of sort */
L1:  /* pass seeking largest key */
       if  A > B  then
```

```
                begin  D = "0";
                       apply  (e); /* exchange A, B */
                       if  C1 > 1  then  apply  (aa)
                end
        else  if  C1 > 1  then  apply  (a)
                          else  apply  (f); /* get next item */
        C1 = C1 - 1;
        if  C1 > 0  then goto  L1; /* test end of pass */
        if  D = "1"  then goto  S3; /* test end of sort */
        apply  (f); /* prepare items for next pass */
        D = "1"; /* set detector for next pass */
S2:     C1 = C = C - 1; P = P + 1; /* update counters */
L2:  /* pass seeking smallest key */
        if  A < B  then
                begin  D = "0";
                       apply  (b);  /* exchange A, B */
                       if  C1 > 1  then  apply  (ff)
                end
        else  if  C1 > 1  then  apply  (f)
                          else  apply  (a)
        C1 = C1 - 1;
        if  C1 > 0  then goto  L2; /* test end of pass */
        /* test end of sort */
        if  D = "0"  then
                    begin  D = "1"; /* set detector */
```

```
                    apply  (a); /* prepare items */
                    goto  S1  /* next pass */
                end;
S3:  /* move ordered items to final locations in data loop */
      if  (P odd)  then  apply  (a)  (P + 3)/2  times
                   else  apply  (f)  P/2  times
end;   /* of  Algorithm B.3  */
```

Appendix C: ALGORITHMS FOR PROJECTION IN ORGANIZATION I.

Algorithm C.1: Performed in each super module (phase 1).

This algorithm is performed concurrently in all super modules containing Relation.

```
Start: C1 = L-1; /* C1 holds number of tuples to be examined */
       C2 = 0; /* C2 holds number of tuples examined in a pass*/
       C3 = 0; /* C3 holds number of duplicate tuples */
       P = 0; /* P holds the number of passes, equal to number*/
              /* of distinct tuples at end of algorithm       */
       read  tuple t1; /* first tuple */
Step 1:   P = P + 1; /* increase number of passes */
Step 1.1: if  (mark(t1) = 01) and  C1 ≥ 1    /* duplicate */
          then  begin  read  tuple t1; /* next tuple */
                       C1 = C1 - 1;
                       goto  step 1.1
                end
          else  if  mark(t1) = 00 then  mark(t1) = 11;/* not a*/
                                                /* duplicate*/
          if  C1 ≤ 0  /* all tuples examin. against each other*/
          then  end of algorithm;
Step 2:   read tuple t2; /*t2 is tuple to examin. against t1 */
          C2 = C2 + 1;
          if  mark(t2) = 00  /* not a duplicate */
          then  begin  compare t1, t2, on Domainset;
```

```
                    if  t2 is duplicate of t1
                    then  begin  C3 = C3 + 1;
                                 mark(t2) = 01
                          end
               end;
          if  P + C3 ≥ L  then  end of algorithm;
          if  C2 < C1  /* not end of pass */
          then  goto  step 2
          else  /* end of pass */
          begin  C2 = 0;       /* initialize counters for */
                 C1 = C1 - 1; /* next pass                */
                 t1 = t2;/* t2 is first tuple for next pass*/
                 reverse shift direction;
                 goto  step 1   /* next pass */
          end;
```

Algorithm C.2: Broadcasting of candidate tuples (phase 2).

This algorithm is composed of two coroutines, one for the selection of the broadcasting super module and the processing in the same, called "Broadcast", and one for the processing in the receiving super modules, called "Receive". Communication between these two coroutines is done through ready/not ready signals.

```
/* SB ≡ {sb| super module sb contains tuples from Relation}   */
/* SR ≡ {sr| super module sr contains tuples from result rel.} */
```

```
Broadcast: SR = ϕ ; /* initially, SR is empty */
Step 1: if  |SB| ≤ 1  /* SB empty or only one super module in SB*/
        then   /* end of algorithm */
        begin  SR = SR + SB;
               signal end of projection operation
        end;
        SB = SB - s1;/* choose super module s1 to broadcast */
        SR = SR + s1;
        C1(s1) = P(s1); /* C1(s1), P(s1) contain the number */
                        /* of candidate tuples in s1        */
Step 2: read tuple t1; /* t1 is a tuple from s1 */
        if  mark(t1) ≠ 11  /* not a candidate */
        then  goto  step 2  /* get next tuple in s1 */
        else  /* t1 is a candidate, result tuple */
        begin C1(s1) = C1(s1) - 1;
              SRC = SB;/* SRC set of receiving super modules */
              set s1 ready for all sb in SB; /* s1 ready */
Step 2.1:     for all super modules in SRC do
              begin if sb ready then
                    begin broadcast t1 to sb;
                          reset s1 ready for sb;
                          SRC = SRC - sb
                    end
              end;
              if SRC not empty then goto  step 2;/* wait */
        end;
```

```
        if  C1(s1) > 0  /* more candidate tuples in s1 */
        then  goto  step 2  /* next candidate from s1 */
        else  /* all candidate tuples broadcast */
        goto  step 1; /* choose another super module */

Receive: set sb ready;
Step 3: if  s1 not ready  then goto  step 3;  /* wait */
        receive tuple t1;
        reset sb ready;
        C1(sb) = P(sb); /* C1(sb), P(sb) contain number of  */
                        /* candidate tuples in sb          */
Step 4: read tuple t(sb);
        if  mark(t(sb)) ≠ 11  /* not a candidate */
        then  goto  step 4  /* next tuple */
        else  /* t(sb) is a candidate tuple */
        begin C1(sb) = C1(sb) - 1;
              compare t1, t(sb), on Domainset;
              if t(sb) is duplicate of t1 then
              begin P(sb) = P(sb) - 1;/* one less candidate */
                    mark(t(sb)) = 01; /* mark duplicate */
                    C1(sb) = 0  /* signal end of search for */
                                /* t1 duplicates           */
              end
        end;
        if C1(sb) > 0  then  goto  step 4; /* next tuple */
```

```
        /* all candidate tuples read or duplicate of t1 found*/
        if P(sb) > 0  then /* sb contains candidate tuples */
        begin shift sb once;              /* prepare for     */
              reverse shift direction;    /* next broadcast  */
              goto  Receive  /* next pass */
        end
        else /* sb does not contain any candidate tuples */
        SB = SB - sb; /* no need to process sb for this  */
                      /* particular projection anymore   */
```

Algorithm C.3: Comparison of tuples from pairs of super modules (phase 2).

```
/* SB ≡{sb| super module sb contains tuples from Relation}  */

Start: determine an optimum set of pairings of super modules in
       SB; /* optimum in the sense of minimum number of      */
           /*     pairings                                   */
       if  |SB| odd  then  C = |SB| /* C contains the current */
                     else  C = |SB| - 1; /* no. of pairings  */
       choose one pairing of super modules; /* any of the above*/
Step 1: for each pair, (s1, s2), P(s1) > 0 and P(s2) > 0, from
            the chosen pairing, do
        /* step 1 is composed of two coroutines */

begin  C1(s1) = C2(s1) = C3(s1) = P(s1); /* initially, all    */
       C1(s2) = C2(s2) = C3(s2) = P(s2); /* counters hold no. */
                                         /* of dupl. tuples   */
```

```
        set s2 ready to receive; /* s2 is ready to receive   */
                                 /*  tuple  from  s1         */
        reset s1 ready;
COR1:   read tuple t1(s1);
        if mark(t1(s1)) ≠ 11  /* not a candidate tuple  */
        then  goto  COR1   /* get next tuple  */
        else
        begin
Step 1.1: C1(s1) = C1(s1) - 1; /* one tuple less for next pass*/
         C2(s1) = C2(s1) - 1; /* one tuple less in this pass */
         set s1 ready; /* s1 is ready to send */
Step 1.2: if s2 is ready to receive
         then  send t1(s1) to s2
         else  goto  step 1.2;  /* wait  */
Step 1.3: if s2 not ready to send then goto  step 1.3  /* wait*/
         else
begin    receive t3(s2);
Step 1.4: read tuple t2(s1);
         if mark(t2(s1)) ≠ 11 and  C1(s1) > 0
         then goto  step 1.4  /* get next tuple */
         else
         begin if C1(s1) = 0 then end of s1 processing;
               C2(s1) = C2(s1) - 1; /* one tuple less */
               if C3(s1) = P(s1) /* no duplicates in this  */
               then                    /*  pass    */
```

```
                begin compare t3, t2, on Domainset;
                      if t2 is duplicate of t3
                      then
                      begin P(s1 = P(s1) - 1;
                            mark(t2(s1)) = 01;
                            C1(s1) = C1(s1) - 1
                      end
                end
          end;
          if C2(s1) > 0  /* more tuples in this pass */
          then goto  step 1.4;
          C3(s1) = P(s1);   /* initialize counters for  */
          C2(s1) = C1(s1);  /*   next  pass             */
end
     end;
     if C1(s1) > 0  and C1(s2) > 0 /* s1, s2 contain tuples to */
     then                          /*   be  compared           */
     begin  reset s1 ready;
            t1(s1) = t2(s1);  /* t2 is first tuple in  */
                              /*   next   pass         */
            reverse shift direction;
            goto  step 1.1
     end
     else  end of s1 processing;
```

```
COR2: read tuple t3(s2);
      if mark(t3(s2)) ≠ 11  and C1(s2) > 0 /* not a candidate */
      then  goto  COR2  /* get next tuple */
      else
      begin if C1(s2) = 0  then goto  step 1.8;
Step 1.5:   C1(s2) = C1(s2) - 1;
            C2(s2) = C2(s2) - 1;
Step 1.6:   if s1 ready then
            begin receive t1(s1);
                  reset s2 ready to receive
            end
            else  goto  step 1.6;  /* wait */
            if C3(s2) = P(s2) /* no duplicates in this pass */
            then
            begin compare t3, t1, on Domainset;
                  if t3 is duplicate of t1
                  then
                  begin P(s2) = P(s2) - 1;
                        C1(s2) = C1(s2) - 1;
                        mark(t3(s2)) = 01;
                        goto  COR2  /* next tuple */
                  end
            end;
            set s2 ready to send;
            send t3(s2) to s1;
```

```
Step 1.7:   read tuple t4(s2);
            if mark(t4(s2)) ≠ 11  and  C1(s2) > 0
            then goto  step 1.7; /* get next tuple */
            if C1(s2) = 0  then goto  step 1.8;
            C2(s2) = C2(s2) - 1; /* one less tuple in pass */
            if C3(s2) = P(s2)  then
            begin compare t4, t1, on Domainset;
                  if t4 is duplicate of t1
                  then
                  begin P(s2) = P(s2) - 1;
                        C1(s2) = C1(s2) - 1;
                        mark(t4(s2)) = 01
                  end
            end;
            if C2(s2) > 0  then goto  step 1.7;
            C2(s2) = C1(s2); C3(s2) = P(s2);
Step 1.8:   if  P(s2) = 0  then  SB = SB - s2;
            set  s2 ready to receive
      end;
      if C1(s2) > 0  and  C1(s1) > 0  then
      begin  t3(s2) = t4(s2);
             reverse shift direction;
             goto  step 1.5
      end
      else  end of s2 processing;
end;  /*  of  step  1  */
```

```
Step 2:  C = C - 1; /* decrease number of pairings */
         if  C = 0  /* no more pairings left */
         then  end of algorithm
         else
         if |SB| < C  or  (|SB| even  and  |SB|  = C)
             /* decrease in  SB  enough to make worthwhile the */
             /* determination of a new set of pairings     */
         then  goto  start
         else
         begin  choose another pairing;
                goto  step 1
         end;
```

Algorithm C.4: Working space approach.

```
/* SB ≡ {sb| super module sb contains tuples from Relation}  */
Start:  apply Algorithm C.1 to super modules containing tuples
        to be examined; /* initially, all super modules      */
                        /* containing tuples from Relation   */
        CONCURRENTLY,
        transfer candidate tuples to empty super modules  in
        working space, filling one super module at a time;
        mark those transferred tuples  00;
        if  sb filled  then  S1 = S1 + sb; /* S1 set of super */
                                           /* modules         */
```

```
Step 2:  if  |S1| ≤ 2  then
         begin  if  |S1| = 1  /* only one super module with */
                              /*   candidate tuples        */
                then  apply Algorithm C.1  /* end of algorithm*/
                else /* two super modules with candidates  */
                apply Algorithm C.3 or C.2;/*end of algorithm */
         end
         else
         if  |SB|  -  |S1| ≥ K  /*  decrease in number of super*/
                                /* modules enough to warrant  */
                                /* another transfer of tuples */
         then  begin  SB = S1;
                      S1 = φ ;
                      goto  start  /* iterate transfer step */
               end
         else  begin  apply Algorithm C.1 to super modules in
                      S1;
                      apply Algorithm C.3 or C.2 to super
                      modules in S1
               end;  /*  end  of  algorithm  */
```

VITA

Mario Jino was born in Quintana, São Paulo, Brazil, on September 11, 1943. He received the "Engenheiro de Eletrônica" degree from Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, São Paulo, Brazil, in 1967 and the "Mestre em Ciencias" degree from Universidade Estadual de Campinas (UNICAMP), Campinas, São Paulo, Brazil, in 1974.

From December 1967 to January 1971 he was a research assistant in the Instituto Nacional de Pesquisas Espaciais, São José dos Campos, São Paulo, Brazil. Since February 1971 he has been an assistant professor in the Departamento de Engenharia Elétrica/Eletrônica, Faculdade de Engenharia de Campinas, Universidade Estadual de Campinas, São Paulo, Brazil.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-78-900	2.	3. Recipient's Accession No.
4. Title and Subtitle: Intelligent Magnetic Bubble Meories			5. Report Date: May 1978 6.
7. Author(s): Mario Jino			8. Performing Organization Rept. No.
9. Performing Organization Name and Address: Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. MCS-77-27910
12. Sponsoring Organization Name and Address: National Science Foundation Washington, D.C.			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

In this paper, we are concerned with the design of intelligent magnetic bbble memories. It is our intent to explore ways of incorporating the novel bubble (ip organizations and bubble movement operations in the design of such memories. In rticular, we evaluate the performance of various file processing algorithms and mory organizations which can be achieved through their use. Retrieval times per rd and per page are the parameters used to evaluate the different memory organi-tions. Performance of hierarchical memory systems using bubble memories is dis-ssed. The unique features of magnetic bubble memories are used in designing gorithms for elementary file processing operations such as sorting, merging d clustering and for basic relational algebraic operations.

17. Key Words and Document Analysis. 17a. Descriptors

magnetic bubble memories, intelligent memory, memory organization, merging, clustering, relational data model, relational algebraic operations.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price